GREEDY ALGOs

Raj
Page:
Date: / /

Q. We are given note values in array and a required amount 'X' if we have ∞ notes (money). Find minimum notes to used

X = 388

eg:

1	2	5	10	20	50	100	200	500	1000

APP

1) Take the max element until you can then move to smaller no.

Here, 1000 ✗ 500 ✗ 200 ✓ 100 ✓ 50 ✓ 20 ✓ 10 ✓ 5 ✓ 2 ✓ 1 ✓

req. amt: (388) (388) (188) (88) (38) 18 8 3 1 (0) ✓

Step 1: Sort Array in decending order

```
int x = 388;
int notecount = 0;
for (i=0 to n)
{
    notecount = notecount + (x / arr[i]);
    x = x - ((x / arr[i]) * arr[i]);
}
return notecount
```

no. of Notes × Value of each

total value

LOGIC

Note = 20

amt = 42

Max 20 Rs note can be used Rs

42 | 20 = 2

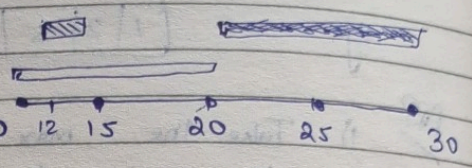
Activity Selection Problem

Page: Raj
Date: / /

Q. given start and end time of "n" activities Select max no. of activities that can be performed by single person, note: Person can only work on single activity at a time.

eg:

start	end
10	20
12	15
20	30



① Sort acc to end time

```
Map <int, int> m;

int take = 0;
int endtime

for (auto it : m)
{
    if (take == 0)
    {
        endtime = it.first;
        take++
    }
else if (it.second >= endtime)
    {
        endtime = it.first;
        take++
    }
}
```

take first as it is

② Compare start time of upcoming with last end time

INPUT

Start (value) end (key)

m[end] = start

this way map will sort acc to end time

Fractional

Q. we have

value / wt. (₹/kg)

we are

Steal the

① Find

② Sort

W = 20

we will

map

Fractional Knapsack

Raj
Page:
Date: / /

Q. We have "n" items with (weight & value) arr[i] in ~~array~~ ~~we need to~~ sack of capacity "W". ~~put items such that~~ value is ~~max~~.

value (₹s)	21	24	12	40	30
weight (kg)	7	4	6	5	6
Value/wt. (₹/kg)	3	6	2	8	5

We are thief with sack capacity $W = 20$ kg.
Steal items ~~of~~ so that we have max value.

① Find value per kg

② Sort in dec order wrt (value/wt.)

W = 20

Value (Rs.)	40	24	30	21	12
Weight (kg)	5	4	6	7	6
Value/wt (₹/kg)	8	6	5	3	2
	✓	✓	✓		

Capacity left: 20 (kg), 15, 11, 5,

here 4 will only take 3 × 5 (kg)

we will store in

map < double (value/wt) , pair < int (value), int (wt) > , greater <> > m

Sorting in dec order wrt (Val/wt)

Page: Raj
Date: / /

Code=

```
map<double, pair<int,int>, greater<>> m;

for (auto i : m)
{       // item wt
    if (i.second.second <= W)
    {
        val += i.second.first; // item value
        W -= i.second.second
    }

    else{
        val += i.first * W
        break;
    }

return val;

}
```

Code-

```
priority_queue<int, vector<int>, greater<>> pq

while (pq.size() > 1) {
    int first = pq.top(); pq.pop();
    int second = pq.top(); pq.pop();

    int mergecost = first + second;
    totalcost += mergecost;
    pq.push(mergecost);
}
return totalcost;
```

Optimal Merge Pattern

Raj
Page:
Date: / /

- You have 'n' files with their computation time in Array
- Choose any two files, add their computation times and append it to the list of computation times {Cost = Sum of comp-time}
- Do this until only 1 file left in array.

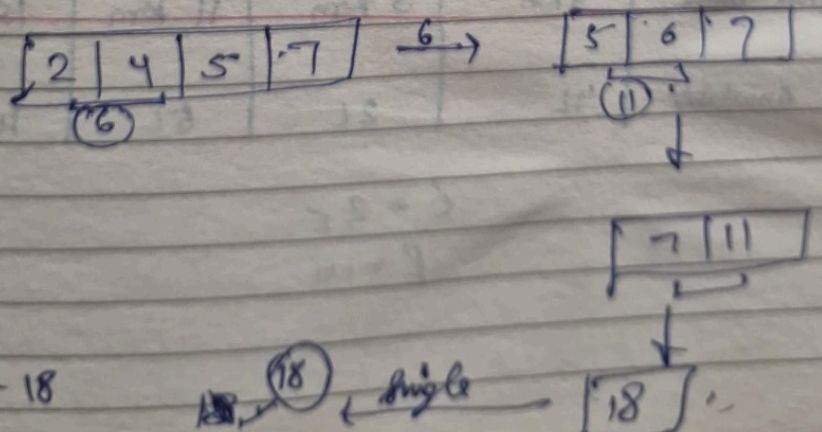
⇒ Do operation such that Cost is minimus after all

eg: [5 | 2 | 4 | 7]

APP

1) Push all element to minheap
2) Take top 2 elements one by one
   - Add cost to answer
   - Push merged file again to Minheap
3) When single element remain
   ↓
   OUTPUT: Cost

Dry Run



Ans) 6 + 11 + 18

EXEDITION. ☆☆☆

Page: Raj
Date: / /

Q. Cow went on trip of jungle in truck

The truck leaks 1 unit fuel every 1 unit of distance

to repair the truck cow need to drive to nearest town ($< 10^8$)
on this road there are "N" fuel stops, we can add fuel. (1 to 100 unit each stop)

☆ Cows want minimum possible stops on way of town

Capacity of tank is large enough to hold any amount of fuel

Initial units of fuel = P
Initial dist from town = L

S1 S-2 S-3 S4

Dist from town	4 km	5 km	11 km	15 km
fuel available	4L	2L	5L	10L

L = 25
P = 10

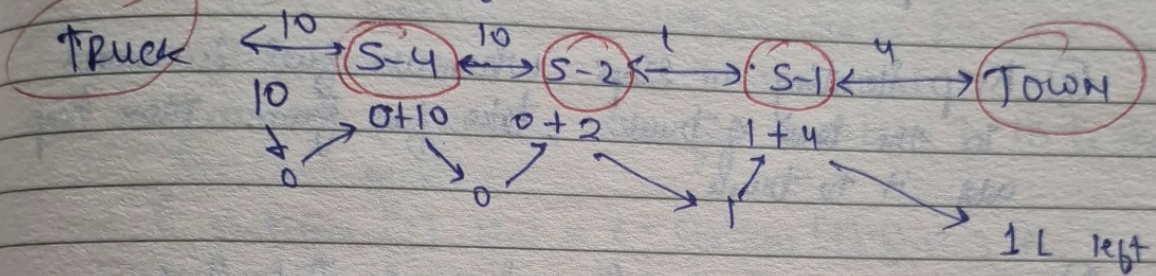
So dist
pofitm

TRue

Raj
Page:
Date: / /

	S-1	S-2	S-3	S-4
So dist from initial position of truck	25-4 = 21	25-5 = 20	25-11 = 14	25-15 = 10

One Possible Way to reach



No. of stop = 3

Page: Raj
Date: / /

optimal

App

① Create a max heap of fuel available at stops

② Sort stops on Basis of distance from initial position

③ Keep iterating on stops whenever fuel is needed in the tank of truck, take fuel from max heap add it to truck

④ Maintain count of stop from which we took fuel.

```
int n;   cin>>n;
vector < pair<int, int> >  stations(n);
// input distance from town and fuel on stops
for (i=0 to n)
{ cin>> stations[i].first >> stations[i].second ; }
int L; cin>>L;          int P; cin >> P
// our dist from town          // current fuel
// convert all stations dist from town based to us to them
for (i=0 to n)
{ stations[i].first = L - stations[i].first ; }
// sort on bases on distance from starting point
Sort (Station.begin() , stations.end());
int stops=0, curfuel=P ; i=0;
// Priority-queue <int> maxheap;
while (curfuel < L)
{
// push all reachable stops in max heap
  while ( i<n && stations[i].first <= curfuel)
  {
     maxheap.push ( station[i].second);
     i++
  }

if ( maxheap.empty() )  return -1

curfuel += maxheap.top()
  maxheap.pop();
 stop++;
```

Max & Min Array diff

Page: Raj
Date: / /

You are given array of "n" size. Remove exactly (n/2) elements from array 1 and put into another

Find max & min values of diff b/t these 2 arrays

$$\left\{ Diff = \Sigma abs\,(A[i] - B[i]) \right\}$$

eg: Input
12, -3, 10, 0

Max diff
A = -3, 0
B = 12, 10
$|\Sigma(A-B)| = 22 - (-3) = 25$

15 10 = 25
|| ||

Min diff : A : -3, 12
B : 0, 10

$\Sigma|A-B) = |0-10| = 5$
$= 3+2 = 5$

Maximise

for sorted array

Max diff = $(A[n-1] - A[0])$
$(A[n-2] - A[1])$
:

$\Phi = [A[n-1] + A[n-2] \cdots A[n/2]$
$- A[0] + A[1] \cdots A[n/2-1]]$

Σ Half elems $- \Sigma$ other Half

Minimise

for sorted array

Count Good no.

Page: Raj
Date: / /

↳ In digit string : Even index have even no.
Odd index have prime no.

Input: n : determine all good no. of length n

Valid Parenthesis

Page: Raj
Date: / /

Q In string having '(' ')' '*' where '*' can be '(' or ')' or empty string
Delimine if string is valid

CODE

```
for(i=0 to s.size())
{
  if(s[i] == '(')
  {
    low++;
    high++;
  }
  else if(s[i] == ')')
  {
    if(low>0) low--;
    high--;
  }
  else
  {
    if(low>0) low--;  high++;
  }
  If At any point high<0
    ↳ FALSE
  if(high<0) return false
}
return low==0
```

APP

* =) or (

- low donate mini open bracket till 'i'
- High donate max open bracket
  * = (
  (to get max open bra)

Only reduce low when you have open brackets

low > 0

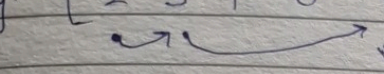
If low == 0, we can Balance all parenthesis

Jump Game - I

Raj
Page:
Date: / /

Q Given array containing max jump length ;
You are at 0th index, need to reach last
So determine if you can do that

eg: [2 3 1 0 4]



eg: [3 2 1 0 4] ✗

APP

If we ever reached Zero (with max jumps)
Then you can't go further

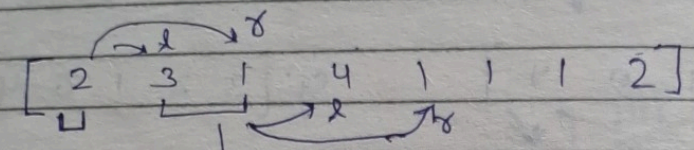
- Traverse each index (by i)
- Maintain a maxInd (max jump at each index)
- If we encounter Zero then FALSE
  Or if i > maxInd then it'll false (means ~~are~~

```
for (i = 0 to n-1)
{
    if ( i > maxInd ) return false
    maxInd = max ( maxInd , i + arr[i] ) ;
}
return true ;
```

Jump Game 2

Page: Raj
Date: / /

Now determine no. of jumps required to reach end.



- max jump = r
- mini jump = l

In this range
Max jump = 3
min jump = 1

Similarly determine farthest jump in l to r range

```
int l=0, r=0, jump=0;
while (r < n-1)
{
    int farthest = 0;
    for (i=l to r)
    {
        farthest = max (farthest, i + nums[i]);
    }
    l = r+1;
    r = farthest
    jump++;
}
return jump;
```

determine Max jump in 'l' to 'r' range

eg: [
7
6
5
4
3
2
1
0

CANDY

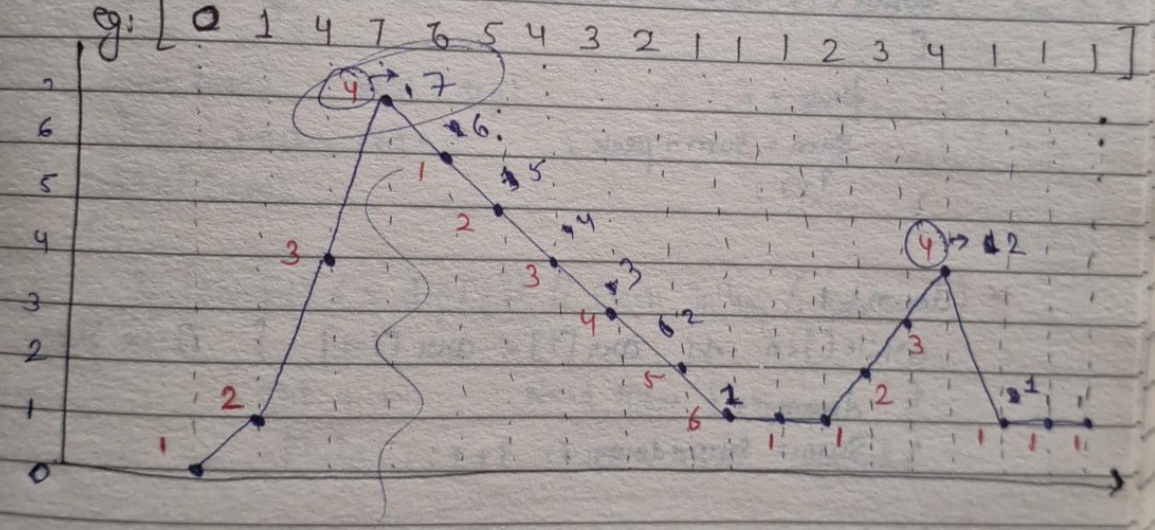
Optimal sol^n = SLOPE.

Page: Raj
Date: / /

n children with their ratings, • give atleast one candy to each
• children with higher rating gets more candies than their neighbours

find min candies required

Input: rating : [1,0,2]
Output : 5
{2, 1, 2} → candies distribution



Sum all these candies to get = Total candies.

peak = 4
down = 7

```
if (down => peak)
{
    sum = sum + (down - peak)
```

Since it would be

Page: Raj
Date: / /

```
sum = 1 , i = 1

while (i<n)
{
  if ( arr[i] == arr[i-1] )
  {
     sum = sum + 1;
     i++;
     continue;
  }
* peak = 1;
  while(i<n && arr[i] > arr[i-1]) // +ve slope
  {
     peak ++;
     sum = sum + peak;
     i++;
  }

* down = 1 //
  while(i<n && arr[i] < arr[i-1] ) // -ve slope
  { down ++;
  } sum = sum + down ; i++;

  if( down > peak)
  {
     sum = sum + (down - peak);
  }
}
```

Flat graph

Min.

Ceiv 2
Determin
there

No

eg: arr
dep

(1) .[S

dep

So
i=
whi
{
i

el

m

}

retu

Min. Platform req.

Raj
Page:
Date: / /

Given 2 arrays with arrival and departure time of trains determine min no. of platform required so that there is not intersection

No of intersection = Platform required.

eg: arr : [9:00 , 9:40 , 9:50 , 11:00 , 15:00 , 18:00]
dep : [9:10 , 12:00 , 11:20 , 11:30 , 19:00 , 20:00]

(1) • Sort arrival and departure arrays

dep: [9:10 , 11:20 , 11:30 , 12:00 , 19:00 , 20:00]

(2) 2 pointer comparison of max clashes b/t trains

```
Sort(arr); Sort(dep)
i=0, j=0, cnt=0
while(i<N)
{
  if(arr[i] <= dep[j]) // Clash. in any 2 time.
  {
    cnt++          Need more Platform
    i=i+1;         check next arrival.
  }
  else{
    cnt--; j=j+1;  // lower the counter bcuz
                   that train already left
  }
  maxcnt = max(maxcnt, cnt);
}
return maxcnt.
```