Constructing Basestations

27th July 2016 Advance Problem

Ranjith Kumar

Problem description

Four 5G base station towers needs to be installed in a Landscape which is divided as hexagon cells as shown in Fig below, which also contains number of people living in each cell. Need to find four cells to install the 5G towers which can cover maximum number of people combining all four cells, with below conditions

- Only one tower can be placed in a cell
- Each of the four chosen cell should be neighbor to atleast one of the remaining 3 cells.
- All four cells should be connected (like one island)

Refer next slide for some valid combinations

Input range: 1 <= N, M <= 15

Sample input Format for Fig in right

3 4

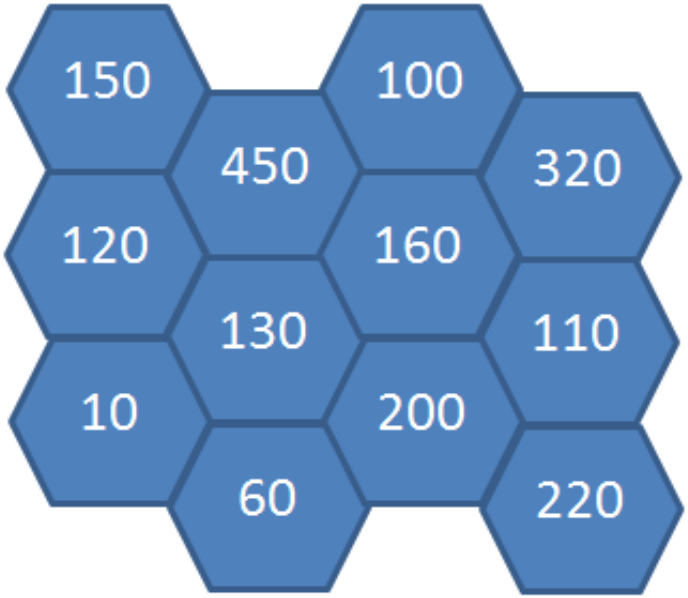
150 450 100 320

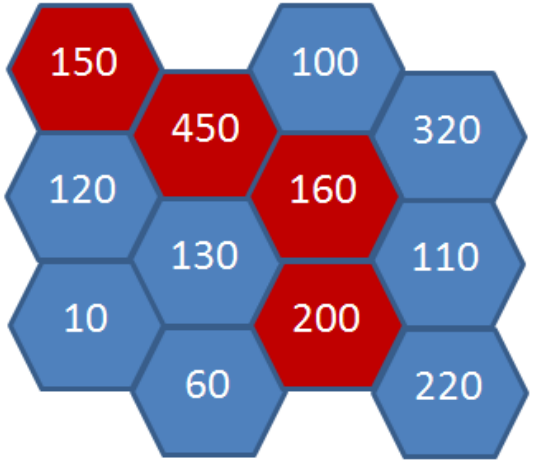
120 130 160 110

10 60 200 220

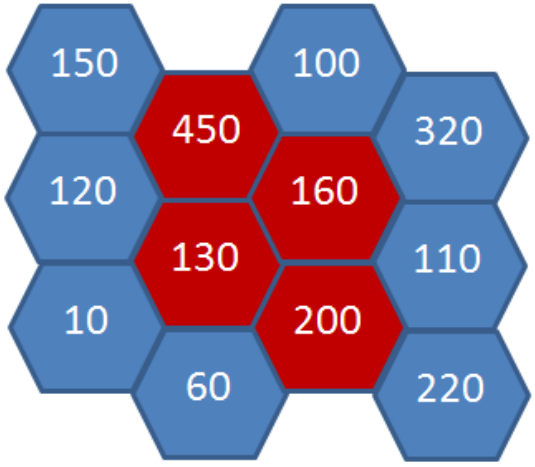
Output

Square of Maximum number of people covered by 4 towers

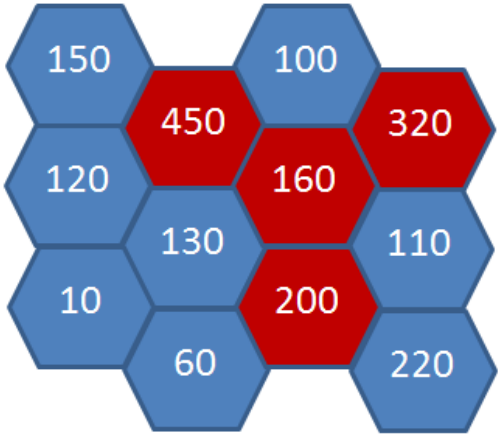




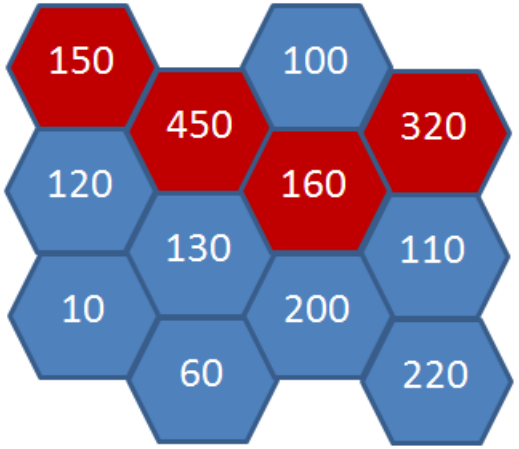
Case 1: sum 960



Case 2: sum 940



Case 3: sum 1130



Case 4: sum 1080

Case 3 has maximum sum, so output is 1130 * 1130 => 1276900

Solutions

- Approach 1:
  - Get logic to find neighbor cells for odd and even cell (w.r.t column)
  - For each cell, do
    - DFS of depth 4
    - combination for remaining number of cells with current cell's neighbor cells only
- Approach 2:
  - Read the input in hexagon format. Get logic to find neighbor cells. In this case logic will be same for both odd and even cell.
  - For each cell, do
    - DFS of depth 4
    - combination for remaining number of cells with current cell's neighbor cells only.
- Approach 3:
  - For each cell give unique number 1, 2, 3, … m*n
  - Generate combination of four numbers from this set and check if these four cells are neighbours.
- Approach 4: (Given by Bhargav Madishetty)
  - Get logic to find neighbor cells for odd and even cell (w.r.t column)
  - For each cell, do
    - DFS of depth 4
    - Calculate Y as shown in figure 3 and also inverted Y.
- Solutions attached for Approach 1 and 2 in Basestation.c, Approach 4 in hexagon.cpp


basestations.c


Hexagon.cpp


input.txt

Common mistakes

- Used same logic to find neighbour cells without differentiating for odd and even cells.
- Some used row index to check even/odd instead of column index.
- Used only DFS of depth 4 to get the combination, missed the combination which includes more than 2 neighbours of current cell as in case 3 in slide 3.

Similar problem in Sotong (Special Outing)

http://sotong.sec.samsung.net/sotong/cp/cpContestMain.do?contestId=AVYw32R1QwvVldFY

Fishing Problem

10th August 2016 Advance Problem

Arun Mahajan

Problem description

Given:

Fishing Spots: 1 to N

3 Gates with gate position and number of fishermen waiting to get in

Distance between consecutive spots = distance between gate and nearest spot = 1 m

Fishermen are waiting at the gates to get in and occupy nearest fishing spot. Only 1 gate can be opened at a time and all fishermen of that gate must occupy spots before next gate is open.

There could be 2 spots closest to the gate. Assign only 1 spot to the last fisherman in such a way that we get minimum walking distance. For rest of the fishermen, ignore and assign any one.

Write a program to return sum of minimum distance need to walk for fishermen.

Distance is calculated as gate to nearest spot + nearest spot to closest vacant spot.

If the gate is at position 4, then fishermen occupying spot 4 will walk 1 m, fishermen occupying spot 3 or 5 will walk 2 m (1m for gate to spot#4 + 1M for spot #4 to spot #3 or 5).

Ex: 3 gates at position 4,6 and 10. Total fishing spots = 10

If gates are opened in order G1->G2->G3

After G1 gate is opened, fishermen are placed at following spots.
Distance = 11m

G1

After G2 gate is opened, fishermen are placed at following spots.
Distance = 5m

G1 G2

After G3 gate is opened, fishermen are placed at following spots.
Distance = 3m

G1 G2 G3

Total distance in this order : 11 + 5 + 3 = 19

If gates are opened in order G2->G1->G3

Case1 –Last fisherman of gate#2 is placed at pos # 7

After G2 gate is opened, fishermen are placed at following spots.

Distance = 3m

G2

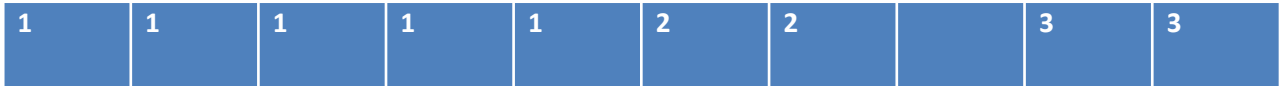
After G1 gate is opened, fishermen are placed at following spots.

Distance = 12m

G1 G2

After G3 gate is opened, fishermen are placed at following spots.

Distance = 3m



G1 G2 G3

Total distance in this order : 3+12+3 = 18

If gates are opened in order G2->G1->G3

Case2 –Last fisherman of gate#2 is placed at pos # 5

After G2 gate is opened, fishermen are placed at following spots.

Distance = 3m

G2

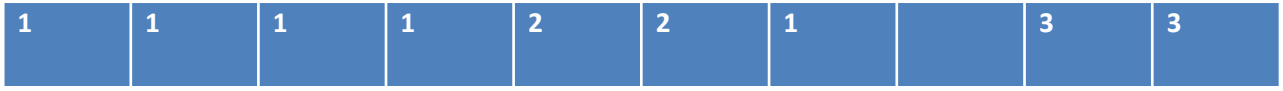
After G1 gate is opened, fishermen are placed at following spots.

Distance = 14m

G1 G2

After G3 gate is opened, fishermen are placed at following spots.

Distance = 3m



G1 G2 G3

Total distance in this order : 3+14+3 = 20

Solutions

- Write function which takes gate # as input and assigns fishermen to nearest spots for that gate. It returns minimum distance and total number of position possible for last fishermen. If number of positions are 2, returns both positions.
- Generate all combinations and assigns fishermen in all gate combinations to calculate minimum walking distance.

- Generating combination can be done in both recursive and iterative way.


fishery_iterative.c


fishery_recursive.c


input.txt


output.txt

Marathon

22nd June 2016 Advance Problem

Ankit Tandon
Abhishek Chaturvedi

Marathon

- Mr. Choi has to do a marathon of D distance. He can run at 5 different paces, each pace will have its time consumed per km and its energy consumption.
- Mr. Choi can only run till he had energy left.
- Find the minimum time required for choi to complete marathon if he has H energy.

Sotong

Approaches

- Using For loop to calculate all combination
- Using recursion with Pruning to find all combinations
- Using Recursion with for and While loops to find all combinations
- Using DP to find the solution (more programming required in this approach)
- Using recursion with memorization
- Please find attached solutions for first 4 approaches

22_June_Solutions(1-2-3).cpp

DpSolution.cpp


22_june_Input.txt

Errors/Bugs

- Calculating all permutations instead of combinations (In recursion)
- Not returning at the base conditions

New Research Center for Rare Elements

20th July 2016 Advance Problem

Nishant

New Research Center for Rare Elements

- Samsung wants to explore some of the rare elements for its semiconductor manufacturing. Scientists use one vehicle to explore the region in order to find the rare elements. The vehicle can move only in explored region where roads have already been constructed. The vehicle cannot move on unexplored region where roads are not there. In the current situation, rare elements are present in explored region only. Unexplored regions do not contain any rare elements.

- Square region is provided for exploration. Roads are represented by 1 and where roads are not present that area is represented by 0. Rare elements will only be on the roads where regions have already been explored. Vehicle can move in four directions – up, down, left and right.

- The shortest path for vehicle to a rare element position is called **Moving Path.** The longest of the paths to all rare elements from a region called **Longest Distance.**

- Scientists need to construct one research center so that the research center will be at the position where the longest path to the rare elements will be shortest. This is called **Shortest Longest Distance.**

New Research Center for Rare Elements

- **Refer the example below:**
- Example:

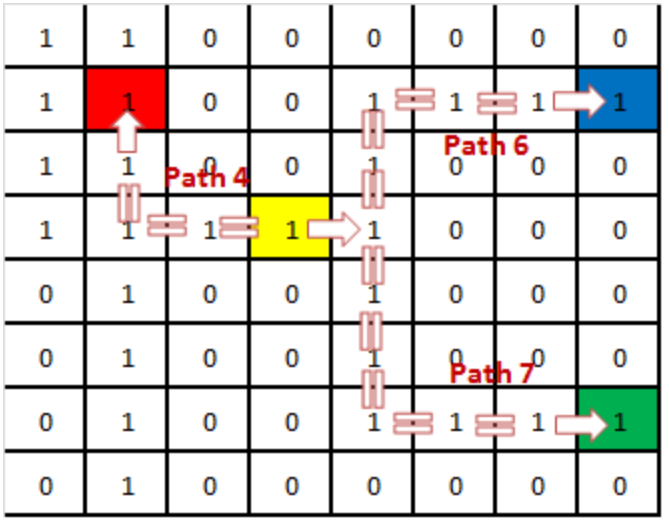


Fig. 1

- In the above picture (Fig. 1), Red, Blue and Green area represents Rare Element area. (2, 2) is represented as Red, (2, 8) is represented as Blue and (7, 8) is represented as Green. So there are three rare elements.

- **If research center is constructed at (4, 4) then distance to Red rare element will be 4, distance to Blue rare element will be 6 and distance to Green rare element will be 7. So the Longest distance will be 7.**

New Research Center for Rare Elements

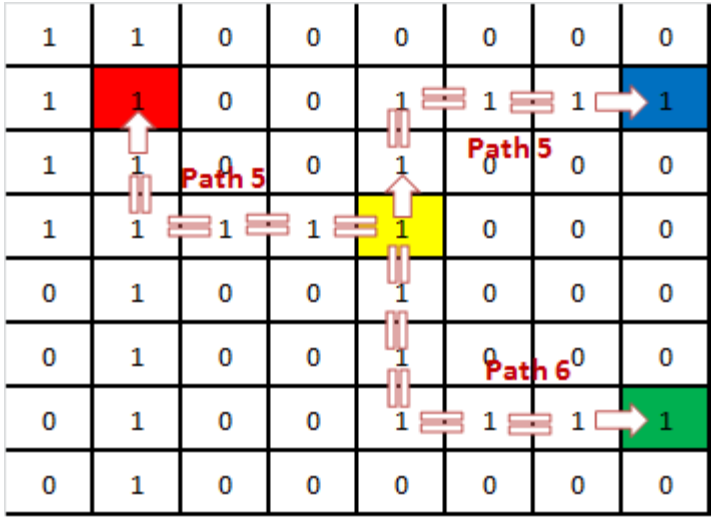


Fig. 2

- Now using the same region (Fig. 2), **if research center is constructed at (4, 5) then distance to Red rare element will be 5, distance to Blue rare element will be 5 and distance to Green rare element will be 6. So the Longest distance will be 6.**
- **So when research center is constructed at (4, 5) then the longest distance will be shortest. And the value of the Shortest Longest Distance will be 6. This will be the output.**
- There can be multiple locations from where the shortest longest distance can be same**. For example if research center is constructed at (5, 5) then still the Shortest Longest distance will be 6.**
- So write a program to find the **Shortest Longest Distance.**

New Research Center for Rare Elements

Constraints:

- The region provided will be square region i.e. NxN (where 5 <= N <= 20).
- There can be minimum of 2 rare elements and maximum of 4 rare elements, i.e. 2 <= C <= 4.
- Roads are represented by 1 while no road area is represented by 0.
- Vehicle can move only on roads in explored area.
- The rare elements will only be present where road are there. Rare elements will not be present where roads are not present.
- Vehicle can move in UP, DOWN, LEFT and RIGHT directions.
- The starting index for rare element is considers as 1.
-

Input:

- First line will be the number of test cases. Second line will indicate region area (N) and number of rare elements (C). Next C lines will contain the position of rare elements. After that N lines will provide the region details where to tell where roads are present and where roads are not present.

Output:

- Output #testcase followed by space and then shortest longest distance.

Sotong

- Laughing Bomb

From test server:

- Human Network – In this problem also BFS can be applied on all the points and then answer can be derived.

Approaches

- Using BFS on each cell to find out the longest path among rare elements from the cell. Then find the smallest in these longest paths. That will provide the solution.
- Few people solved using BFS from rare elements positions. More optimized.

New_Research_Center_for_Rare_Elements.c

input.txt

Errors/Bugs

- Boundary conditions.
- Starting index is 1 in the problem statement.
- Not able to find the longest path among rare elements.
- Improper implementation of queue.

May- 1st exam

Product manufacturing

A company has to produce IOT products of different models,
Each product requires cpus,memories and boards. After production of models, some spare equipments may be left.
In these, cpus and memories can be sold as spare parts but boards cannot be sold.
Due to manufacturing constraints maximum 3 models can be produced.
Each product can be sold at the cost of its model.
Given N different models.
D cpus with price d each.
E memories with price e each.
F boards.

Input:-
T number of testcases, followed by testcases,
Each test case consists of
D total number of cpus available.
E total number of memories available.
F total number of boards available.
N number of models followed by N lines consisting of
a_i, b_i, c_i and p_i where a_i is the number of cpus,b_i number of memories, c_i number of boards
required for producing one unit of that model and p_i is the selling price of the one unit of that model.

Output:-
Print the testcase number followed by the Maximum profit that can be made.

Note:- Maximum profit can also be attained without any production that is by just selling its components.

Constraints:-
$1 \le N \le 8$, $1 \le D,E,F \le 100$, $1 \le d,e \le 10$, $1 \le a_i,b_i,c_i \le 5$, $1 \le p_i \le 100$

Complexity

Simply question is :

we need to choose up to 3 out of 8 (0-3)products,

Answer = Value of products+ remaining CPU*CPU_COST+remaining MEMORY*MEM_COST

Lets try to calculate complexity for brute force approach.

1.We need to choose max 3 out of 8 products, so for that complexity is 8c3=56

2. (taking product-1 out of chosen products from 0 to MAX possible if we take only this product)*(taking product-1 out of chosen products from 0 to MAX possible if we take only this product)*(taking product-1 out of chosen products from 0 to MAX possible if we take only this product) = (100*100*100)

So final complexity is = 8c3*100*100*100 < 10^9.

3. So If complexity is less than 10^9 we can freely go ahead

Input:

7

2 2 2 1 1
1
2 2 2 6

5 10 10 1 1
2
2 1 1 8
1 1 1 6

10 10 10 2 1
1
1 2 2 3

4 6 4 2 1
4
2 4 2 9
1 3 1 7
2 1 1 8
1 2 2 6

40 80 60 1 3
7
3 2 2 56
5 4 2 12
3 5 3 65
1 2 5 78
5 5 2 85
4 2 3 76
5 5 1 48

100 100 100 6 10
8
3 3 1 74
2 3 1 41
3 2 1 64
2 2 3 68
2 2 2 71
2 3 2 66
2 3 3 84
3 3 1 48

1
100 100 100 1 1
8
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

Output:

#1 6
#2 35
#3 30
#4 21
#5 1338
#6 3550
#7 200

Product_BruteForce.java

Time taken including input reding.
#1 time = 0.002
#2 time = 0.002
#3 time = 0.001
#4 time = 0.003
#5 time = 0.064
#6 time = 0.203
#7 time = 0.001

Precautions to take care:

1.Question asked is we need to choose up to 3 out of 8 products, means we can choose no product also and sell all individual components.

2. When choosing a product we need to check whether the component I am spending for this product worth more than if I sell individual products, if not ignore product completely

3.When we apply 3 for loops to generate 8c3 combination, we need to consider N<3 also

4. Always calculate time roughly before selecting approach as explained above.

SW Competency – Stepping Stones to Recursion Approach

Vijay Kumar Mishra
vijay.mishra@samsung.com

Preface

* Further slides compile a list of problems which will incrementally increase test-takers capability to comprehend and attack the problems with recursion approach. These problems, if attacked honestly, will hone the logical thinking of the test-taker into the direction of recursive approach towards problem solving

Pre-Requisites

* These problems expect that the test-taker is familiar with the concept of arrays & graph data-structure. Also test-taker is familiar with recursion and its implementation. These problems in increasing order of difficulty will harness the test-takers capability in various flavors of recursion.
* The test taker has access to soft-tech and so-tong websites hosted in Samsung for software competency practice

Problem List

* *Sky Map*
* *Chess*
* *Finding Matrix*
* *Laughing Bomb*
* *Airfare*
* *Picking Up Jewels*

Sky Map Problem

This Problem intends to test the test takers basic ability to apply ***basic recursion.***

Problem Link :
http://sotong.sec.samsung.net/sotong/practice/practiceProbView.do?practiceProbId=AUFOoOoFE6vVlXvX

The test taker is advised to tackle the problem himself. The below solution reference is provided just for shaping understanding of the concept

Solution :

Chess Problem

This Problem intends to test the test takers basic ability to apply ***basic recursion, pruning and revisiting already visited nodes while performing recursion.***

Problem Link :
http://sotong.sec.samsung.net/sotong/practice/practiceProbView.do?practiceProbId=AUZbHiP1Ro3VldEC

The test taker is advised to tackle the problem himself. The below solution reference is provided just for shaping understanding of the concept

Solution :

Finding Matrix Problem

*This Problem intends to test the test takers basic ability to apply **basic recursion and managing the data within.***

Problem Link :
https://swexpertacademy.samsung.com/common/swea/solvingPractice/problemDetail.do?contestProbId=AVRbA1SfALMAAAHy&problemProcess=1&isFavorite=&probAttack=&_problemLevel=on&_problemLevel=on&_problemLevel=on&_problemLevel=on&ySolveFlag=y&problemTitle=&rowNum=10&pageIndex=1

The test taker is advised to tackle the problem himself. The below solution reference is provided just for shaping understanding of the concept

Solution :

ChemSubstances

Laughing Bomb Problem

This Problem intends to test the test takers basic ability to apply ***basic recursion, forward move compatibility and revisiting already visited nodes.***

Problem Link :

http://sotong.sec.samsung.net/sotong/practice/practiceProbView.do?practiceProbId=AUh3hAH1BFHVldFk#

The test taker is advised to tackle the problem himself. The below solution reference is provided just for shaping understanding of the concept

Solution :

Air Fare Problem

This Problem intends to test the test takers basic ability to apply ***basic recursion and backtracking the same.***

Problem Link :
http://sotong.sec.samsung.net/sotong/practice/practiceProbView.do?practiceProbId=AUdYMRrVANnVldEL#

The test taker is advised to tackle the problem himself. The below solution reference is provided just for shaping understanding of the concept

Solution :

Picking Up Jewels Problem

This Problem intends to test the test takers basic ability to apply ***basic recursion, backtracking, keep the path list concept.***

Problem Link :
http://sotong.sec.samsung.net/sotong/practice/practiceProbView.do?practiceProbId=AUKx1xz1KfvVlXvX

The test taker is advised to tackle the problem himself. The below solution reference is provided just for shaping understanding of the concept

Solution :


Thank you

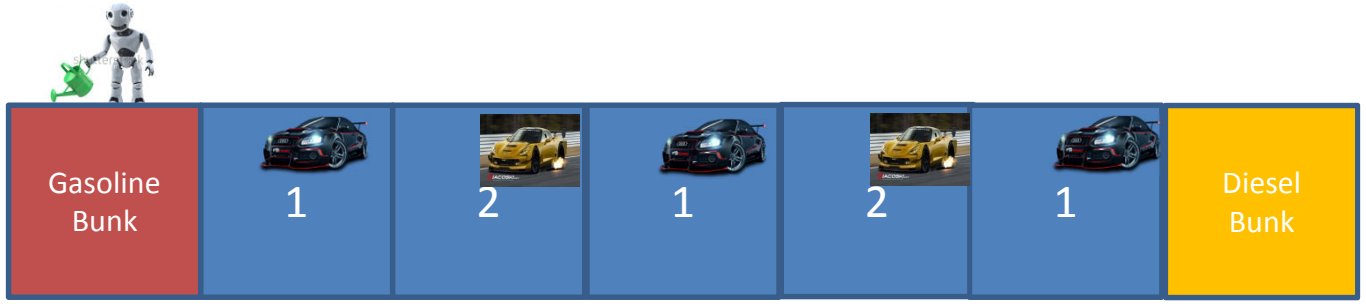
Robot Car Fueling

21-Sep-2016 Advance Problem

Chandru Byadgi & Rohit Bendre

Problem statement:

There are N cars parked in a row in a parking lot of the newly constructed club. as it is demonstrated in the picture below.



There is a gasoline and diesel fueling station installed.at the left and right side of the park.
An automatic fueling robot carries the fuel from station and fill up the parked car with fuel.
The cars are divided into 2 types depending on whether it is a gasoline or diesel car.
1 is denoted as gasoline cars and 2 is denoted as diesel cars.

The automatic robot will be used to provide a cost free fueling service which is filling up all cars with 1 litre of each corresponding fuel.
The robot will move in between the 2 fuelling stations as below :

1) The robot carries 2 litre of gasoline at the gasoline station and starts moving from there.
2) The robot can fill up the cars of the same type of gas it carries 1 litre each.
3) The robot can go back to the fuelling station at any time, Independent from the current amount of fuel it carries.
4) When the robot arrives at the fuelling station, it gets 2 litre of supply of the corresponding fuel.(If the robot has some remaining fuel it will be discarded).

Problem statement:

5) There is an equal distance of 1 between each fueling station and the cars.

The fuel type of N Cars parked in the parking lot will be given.
Find the minimum moving distance of the automated fueling robot after it has filled up all the cars with 1 litre of fuel each.

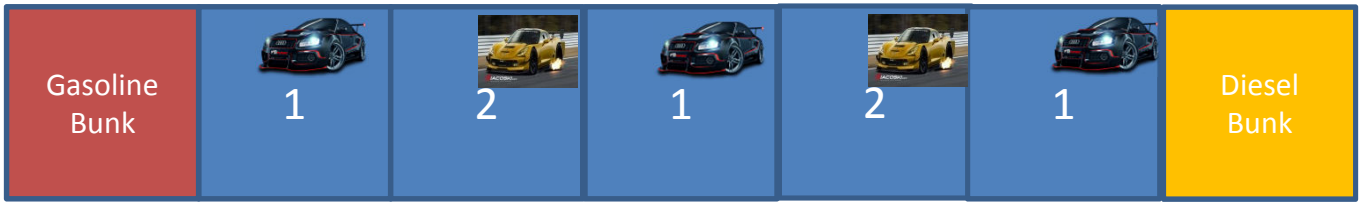
Time limit: C/C++/Java: 3 seconds.
Test cases: 50
2<=N<=8
I/P format:
2 → Total number of test cases
5 → N(Number of cars between gasoline and Diesel stations)
1 2 1 2 1(1→ Gasoline car, 2→Diesel cars)
5
2 1 1 2 1

O/P:
#1 12
#2 14

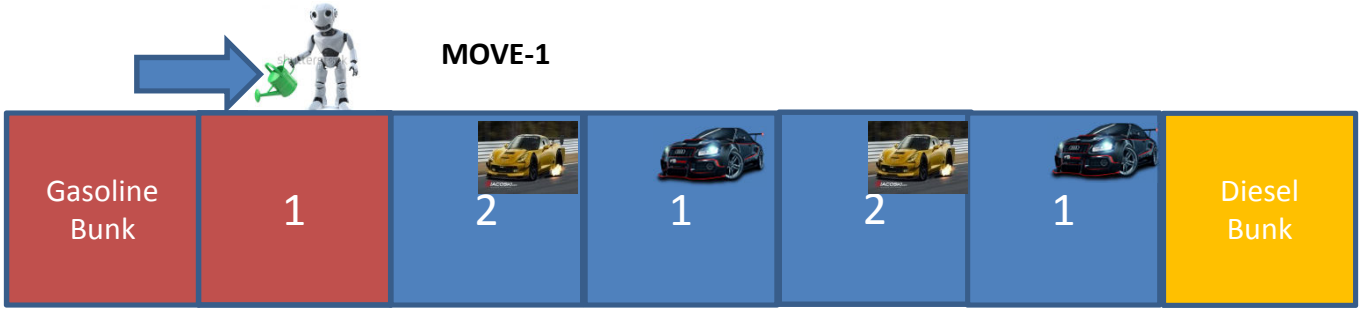
Example 1) Given the total number of cars N = 5 and the order of the parked cars such as G - D - G - D - G (PS: G-> Gasoline, D->Diesel)
the process of finding the minimum moving distance for fueling the car is as follows :

Initial positions of cars and Robot

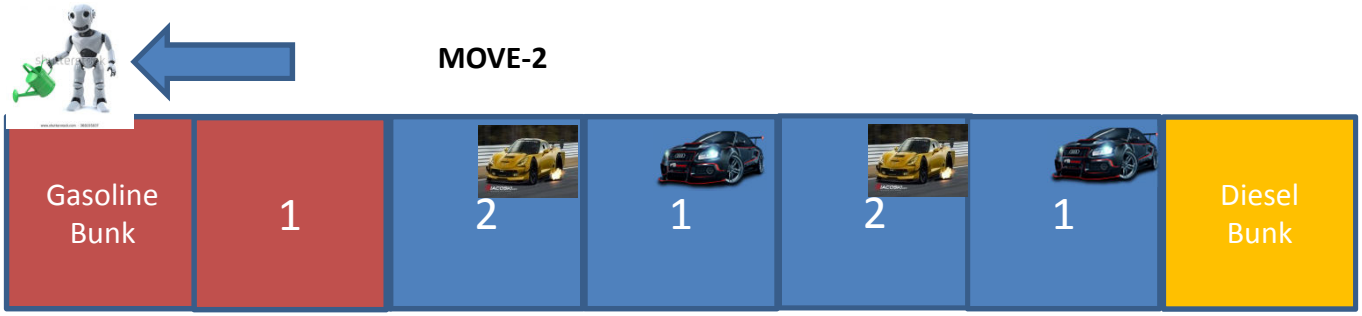
Example:1



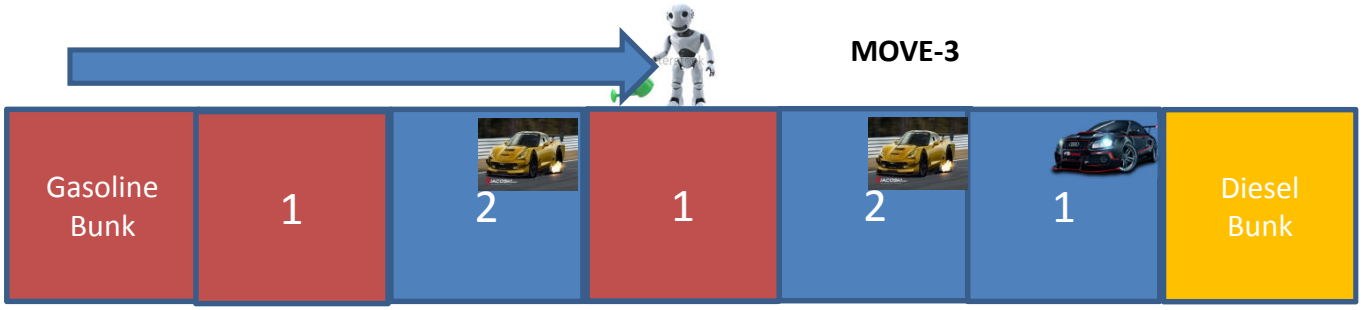
Initial position of cars and robot



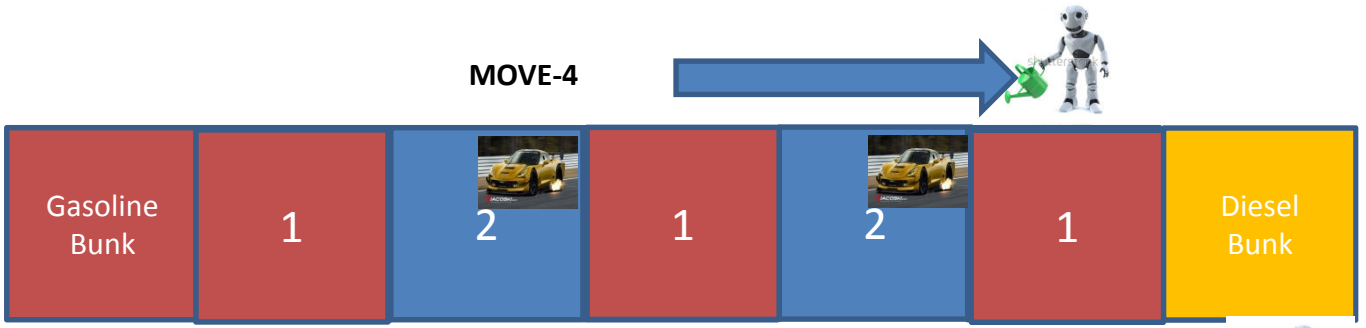
Distance covered **=1** (gasoline bunk to 1st car)



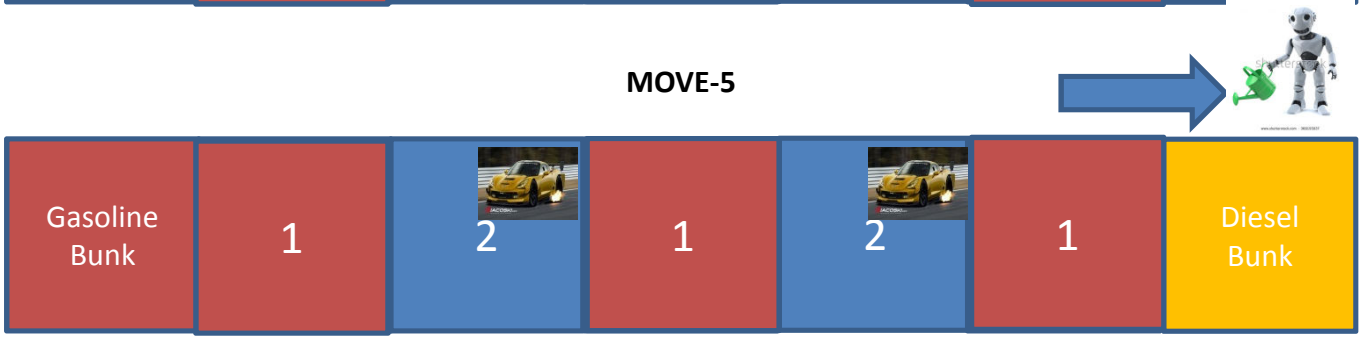
Distance covered **=1+1 = 2** (1st car to Gasoline bunk)



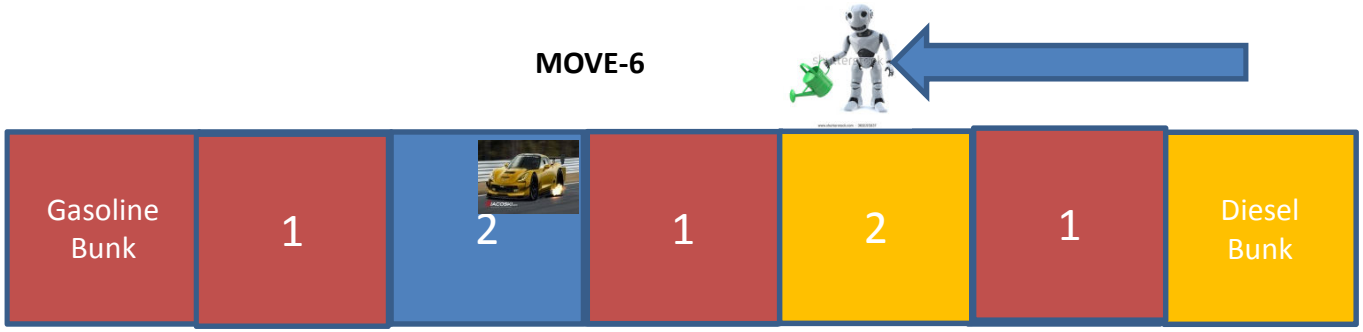
Distance covered **=2+3 =5** (Gasoline bunk to 3rd car)



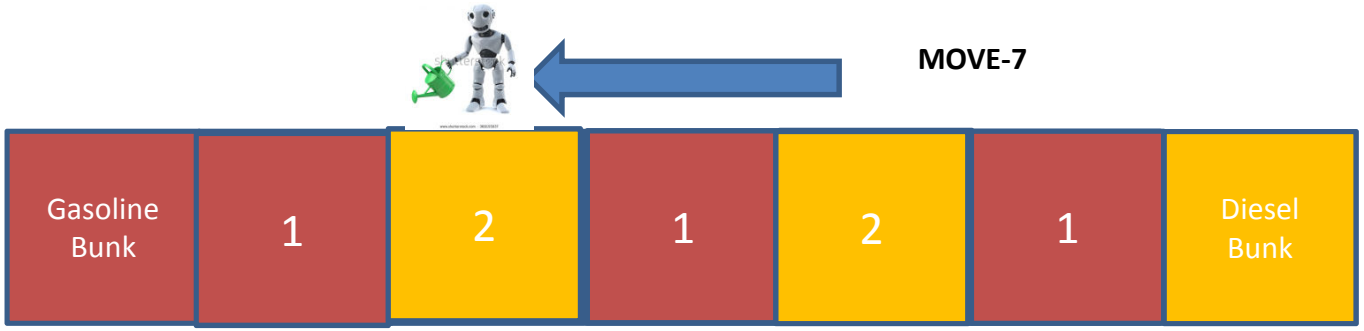
Distance covered **5+2 = 7** (3rd car to 5th car)



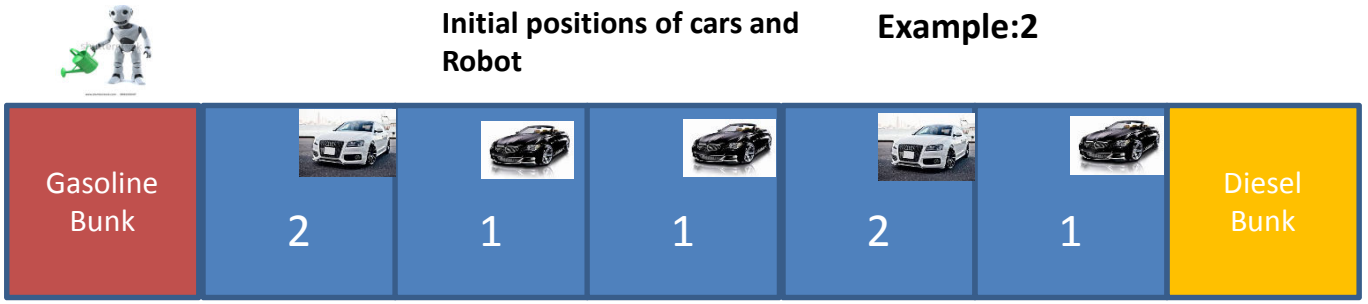
Distance covered **7+1=8** (5th car to Diesel bunk)



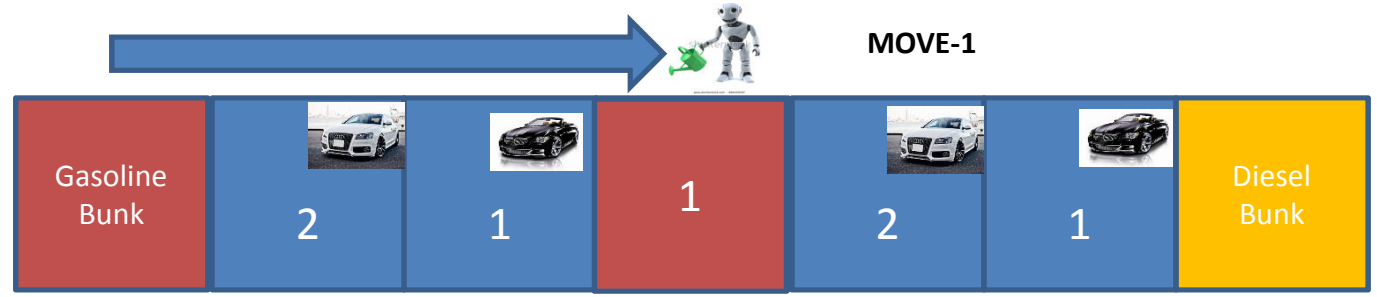
Distance covered **8+2=10** (Diesel bunk to 4th car)



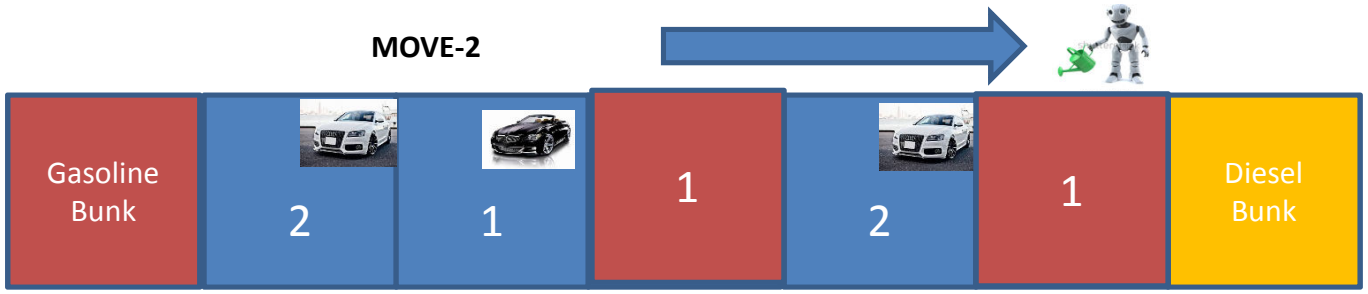
Distance covered **10 +2=12** (4th car to 2nd car). So **12** is the shortest distance among all the available options possible



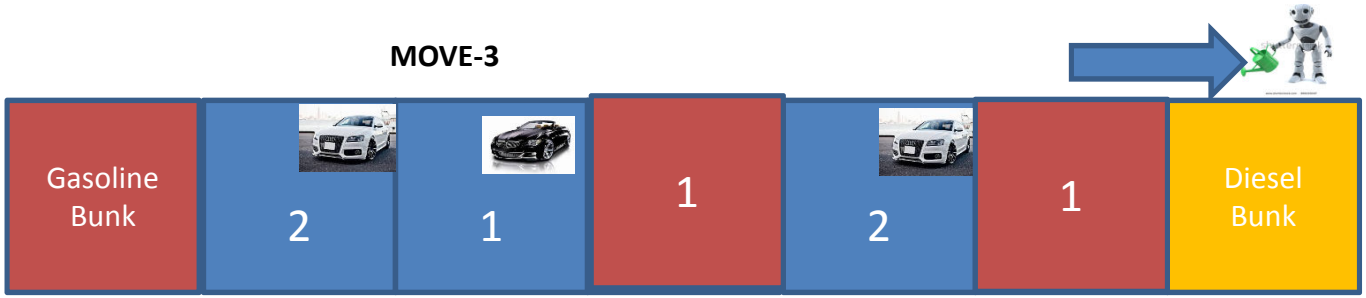
Initial position of cars and robot



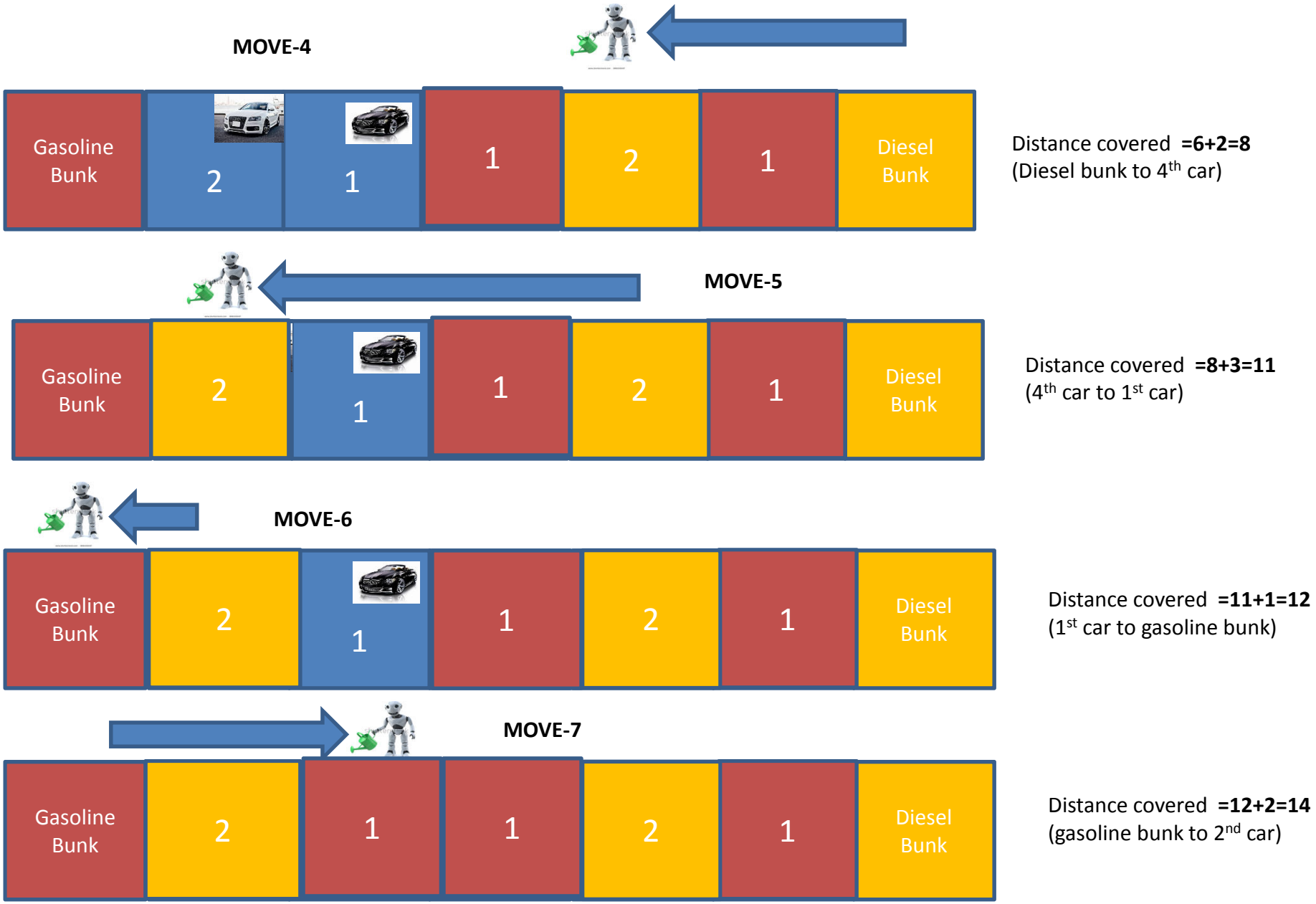
Distance covered **=3** (gasoline bunk to 3rd car)



Distance covered **=3+2=5** (3rd car to 5th car)



Distance covered **=5+1=6** (5th car to Diesel bunk). So **14** is the shortest distance among all the available options possible


MOVE-4
Gasoline Bunk
2
1
1
2
1
Diesel Bunk
Distance covered =6+2=8
(Diesel bunk to 4th car)
MOVE-5
Gasoline Bunk
2
1
1
2
1
Diesel Bunk
Distance covered =8+3=11
(4th car to 1st car)
MOVE-6
Gasoline Bunk
2
1
1
2
1
Diesel Bunk
Distance covered =11+1=12
(1st car to gasoline bunk)
MOVE-7
Gasoline Bunk
2
1
1
2
1
Diesel Bunk
Distance covered =12+2=14
(gasoline bunk to 2nd car)

Approach-1:

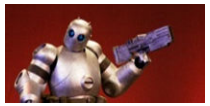
- We should always start from the Gasoline station.
- Once we fuel any car, we have 3 options to perform.
    1) Fuel next car(gasoline or Diesel car, with all the combinations)
    2) Go to Gasoline station and start refueling
    3) Go to Diesel station and start fueling.

Keep updating the distances as we move, once all cars are over, store result in global variable , if we find optimal distance with the current combination than the previous combinations.

Solution is attached:

robo_fueling.cpp

Approach-2:



Robot has Two Functions in the problem statement

Robot @ Pump
Fill 2 Units of Fuel
Move in next direction
Increment count

Robot @ Car
If fuel carried by robot and car not same
increment count
If fuel carried by robot and car is same same
3 decisions
Don’t fill Fuel, continue to next car
Fill the fuel and continue to next car
Fill the fuel and continue backward
Increment count

Pseudo Algorithm

If it is a Pump
    Fill 2 Units of Fuel
    Move in next direction(Gasoline right/Diesel left)
    Increment count
If it is a Car
    If fuel carried by robot and car not same
        increment count
        move next
    If fuel carried by robot and car is same and empty
        if this is last car
            note the count
            return
        Don’t fill Fuel, continue to next spot
        Fill the fuel and continue to next spot
        Fill the fuel and continue backward

Solution is attached:

robo_fueling_rohit.cpp

Rock Climbing

18th May 2016 Advance Problem

Nayan Ostwal

Rock climbing

There is a man who wants to climb a rock from a starting point to the destination point. Given a map of the rock mountain which N = height, M = width. In the map, character '-' is the possible foot place spot (where can climb).

He can freely move up/down at vertical spots which '-' exists sequentially. It's **impossible** to move horizontally in case there is more than one space between '-' in the same height level.

Depending on how high/low he moves towards the upper or lower direction at one time, the level of difficulty of rock climbing gets determined.

The maximum height of moving from the starting point to destination point is the level of difficulty of rock climbing .

The total distance of movement is not important. There are more than one path from the starting point to destination point. => Output: The minimum level of difficulty of all rock climbing paths level.

Hint: Start with difficulty level 0 and then keep increasing it one by one.

Sotong

- The sample test is present in Sotong, CoBY.

- http://sotong.sec.samsung.net/sotong/cp/cpContestProbShow.do?contestId=AVTNeGq1677VldEa&contestProbId=AVTIOFaFGX7VldEa

Approaches

- DFS with recursion
- BFS can also be used with increase in code complexity
- Few people solved it with Backtracking as well (not advised though)
- If the visited array is marked as “steps_count” under consideration, then we need not initialize the visited array with 0 for every “step_count”.
- Attached is DFS solution with TC generation as well as timing calculation

rock_climbing.txt

Errors/Bugs

- Executed DFS from source and destination both at the same “step_count”.
  - This can lead to error as it can now jump 2*step_count.
- Did not change the visited array properly after incrementing the “step_count”
- Tried to jump only the current “step_count” in consideration.
  - You should consider all the steps from 0 to “step_count” in every loop.

Shooting Balloon (Finding Max Score)

27-April-2016 Advance Problem

Sundeep/Nishank

Shooting Balloon (Finding Max Score)

SWC-Advance-Test- 27-April-2016

There will be a N Balloons marked with value Bi (where B(i…N)).
User will be given Gun with N Bullets and user must shot N times.
When any balloon explodes then its adjacent balloons becomes next to each other.
User has to score highest points to get the prize and score starts at 0.
Below is the condition to calculate the score.

1. When Balloon Bi Explodes then score will be a product of Bi-1 & Bi+1 (score = Bi-I * Bi+1).
2. When Balloon Bi Explodes and there is only left Balloon present then score will be Bi-1.
3. When Balloon Bi Explodes and there is only right Balloon present then score will be Bi+1.
4. When Balloon Bi explodes and there is no left and right Balloon present then score will be Bi.

Write a program to score maximum points.
Conditions:

- Execution time limits 3 seconds.
- No of Balloons N, where 1 <= N <= 10
- Bi value of the Balloon 1 <= Bi <= 1000.
- No two Balloons explode at same time.

Input/ Output

Input:
Consists of TC (1 <= TC <= 50).
N – No of Balloons.
B0…..BN N Balloons with their values .
Output:
#TC SCORE

Sample Input:
5
4
1 2 3 4
5
3 10 1 2 5
7
12 48 28 21 67 75 85
8
245 108 162 400 274 358 366 166
10
866 919 840 944 761 895 701 912 848 799

Sample Output:
#1 20
#2 100
#3 16057
#4 561630
#5 6455522

Analysis

1)Aim is to find max score

2) Max score depend on points on neighbor, however there is no easy way to find which sequence which gives max score, so only way is to find the all possible sequence can get max out of it.

3) As order matters in sequence for input N we can have N! sequences, ie. nPn ways (1^{st} balloon N ways, 2^{nd} N-1 ways …last balloon 1 ways N*(N-1)(N-2)..2*1= N!

rock_climbing.txt

Complexity:

- To generate the all sequence O(N!)
- To Get the Score for 1 sequence, for each balloon in sequence we need to left and right neighbors worst case need complete traversal in array so complexity is O(N*N)
- Total complexity is O(N!) * O(N*N) (note: computation has done at end of each sequence)
- 50 TC , N<= 10 => 50 * is O(N! * N*N) => 50 * 100 * 10! => 5000 * 3628800 => 1.5 * 10^10 this cannot be executed in given 3 sec (10^9 instruction per second).
- So need to look for optimization

Pseudo code to generate all sequences.

```
INPUT[N]
CHOICE[N] <= -1 //initialize to -1
Permute(0)
Permute(Position)
{
//stop condition
If( all balloon shot )
{
Compute the score for this sequence in CHOICE[]
If score better than previous then store
}

For i:0~N-1
{
If (ith balloon not selected // CHOICE[i]==-1)
{
Select ith balloon  // CHOICE[Position]= i
Permute (Position+1)
Unselect ith balloon// CHOICE[Position]= -1
}
}
}
```

Optimization

- We can see in above algorithm 2 major operation are carried out 1) generate all sequence O(N!) and 2) computing score for each sequence O(N*N)
- We cannot optimize the algorithm generate all sequences however we can reduce the computing part further.
- Optimization computing part
  - If can optimize the finding the neighbor to O(1) we can reduce computation part to O(N) which leads our algo to execute in 1.5 * 10^9 which can be achieved in 3 sec.
  - Alternatively we can compute the score for each chosen balloon to shoot “on the go” here finding neighbor is extra when each time balloon is chosen which can be O(N) and also reduce 1.5 * 10^9
- If we combine 1 and 2 we can further reduce the time to 1.5 * 10^8

Algorithm to get neighbors

Naïve method by O(N):

```
Neighbor(chosen)
        For Left: chosen-1~0 if Left th balloon not chosen break;
        For Right: chosen+1~N-1 if right th balloon not chosen break;
        if(Right==N)
                Right=-1;
        Return   Left and right ;
```

Optimized way by O(1)

1. Keep 2 array left[] and right[] which contain neighbors of each balloon.
2. Initially neighbor are known, for ith balloon left is i-1 and right is i+1 except that 1st balloon will have no left and last have no right.
3. When balloon is chosen we can obtain its right and left by O(1)
4. When a balloon is shot update neighbor left[i+1]=left[i] right[i-1]=right[i]

Note:

Instead of calling the new function to get left and right calculating left and right inside the recursive faction will reduce many hidden instructions as to call new function compiler add many instruction which can be reduced

Alternative Way

Way to compute the score on the go

- Pass current score variable to recursive function
- When a balloon is chosen to shoot get the left and right neighbors
- Compute the score gained by shooting chosen balloon
- Add this to given score and pass to next level

```
Permute(Position, score)
{
//stop condition
If( all balloon shot )
{
If score better than previous then store
}

For i:0~N-1
{
If (ith balloon not selected // CHOICE[i]==-1)
{
Select ith balloon  // CHOICE[Position]= i
Gain = Compute the gain by shooting ith balloon
Permute (Position+1, score+ Gain)
Unselect ith balloon// CHOICE[Position]= -1
}
}
}
```

Errors/Bugs

- Error in algorithm to generate permutation
- Not optimizing the Computing the score algorithm.
- Stop condition in recursive problem
- Selecting greedy methods

Alternative optimized approach(Divide and Conquer) and Dynamic programming

The problem at first doesn't seem like a divide and conquer problem.

- Reason: If we select a balloon(for bursting) then our array would be divided into two sub arrays. But these two sub arrays won't be independent sub problems.
  - Example
    - Consider 5 balloons B1,.., B5. Bursting B3 divides the array into two sub-array {B1, B2} and {B4, B5}. But these two sub array are not independent of each other ie. score for bursting B4 is dependent on bursting order of {B1, B2}.

B1	**B2**	**X**	**B4**	**B5**

Key Insight

- To divide the problem into two halves we have to ensure that any action(bursting of balloon) in one half doesn't affect score of the other half.
- If we fix a balloon and ensure that we won't burst it until we burst all the balloons to the left of it and all the balloon to the right of it then we can successfully divide the problem into two sub-problems.
- Example
    - Consider the previous case of five balloons. Now instead of bursting B3 we fix that we will burst B3 after all the balloons this makes {B1, B2} and {B4, B5} independent of each other ie score for bursting B4 is now independent of {B1, B2}.

- Another way to visualize the divide and conquer approach is that we think of the problem in reverse. The parallel problem would be given a set of n deflated balloons each with a score, choose the order in which you will inflate the balloon. The score for inflating the balloon is equal to product of score attached to the balloons located left and right to the mentioned balloon.

Pseudo Code

- Note:
  - We store the the input score values in the array inp_arr[N+2].
  - The values corresponding to the ith baloon is store at inp_arr[i].
  - inp_arr[0] = inp_arr[N+1] = 1;

```
getMaxScore(inp_arr, left_limit, right_limit, N){
        initialize max_score = 0;   //Max Score Value to Be Returned
        for(i: left+1 to right-1){
                initialize curr_score = 0;
                curr_score = getMaxScore(inp_arr, left, i, N) + getMaxScore(inp_arr, i, right,
                if(left == 0 && right == N){
                        curr_score += inp_arr[i];
                }
                else{
                        curr_score += inp_arr[left]*inp_arr[right];
                }
                //Update max_score value
                if(curr_score> max_score){
                        max_score = curr_score;
                }
        }
        return max_score;
}
```

The above problem can be easily optimized to include memoization using 2 Dimensional DP Matrix.

Execution time for different approach

For input given in this document.

Generating sequences and computing at end by list for finding neighbor

Execution time: 0.934000 seconds.

On the way compute

- external call for finding neighbor : Execution time: 1.223000 seconds.
- Inline for finding neighbor: Execution time: 0.657000 seconds.
- List for finding neighbor: Execution time: 0.616000 seconds.

Divide and conquer:

- Execution time: 0.004000 seconds.
- with DP: Execution time: 0.001000 seconds.

SE SW Competency Task Force(SSCTF)

Class - 4

SSC Task Force

Anju Bala

Arabinda Verma

Chetan Sai Kumar T

Prathap HS

Vikas Agrawal

Veeraj S Khokale

Sasikumar Sathasivam

- # Toll Gate Problem
  - **(Similar to Mar'16 Adv Test)**
  - **Brainstorming session and approach to resolve**

Class-4

Practice: Advance Test –TollGate–Mar'16

Please find the **minimum cost** to travel from Source to Destination location with multiple toll gates across
There are challenges at each toll gate to minimize the cost.

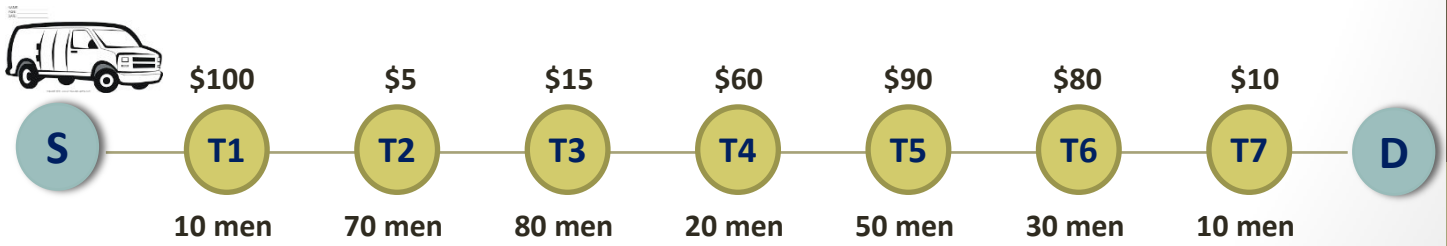
- One can either choose to **pay the toll** or
- One can battle at the toll gate to **avoid paying** by having his own set of men's they travel with them (initially zero) or
- One can **pay double the toll cost and hire all the men** at the each tolls for the next toll to battle and avoid toll cost,
- If you choose to battle at particular toll only if you can have no.of.. hired men is more than the count hired men at respective toll gate.

Note: Each hired men can battle for 3 times only

- For each battle you will lose equal no.of.men with you as well as available in the toll gate . Rest of them will lose 1 round of battle irrespective of they are alive or not. After 3 battle they will not survive. *If you have 10 men with you and toll no. of. Toll men is 8, then you lose 8 men in battle and remaining 2 men lost 1 round of battle and hence they can be available for 2 more rounds only.*

Ex:

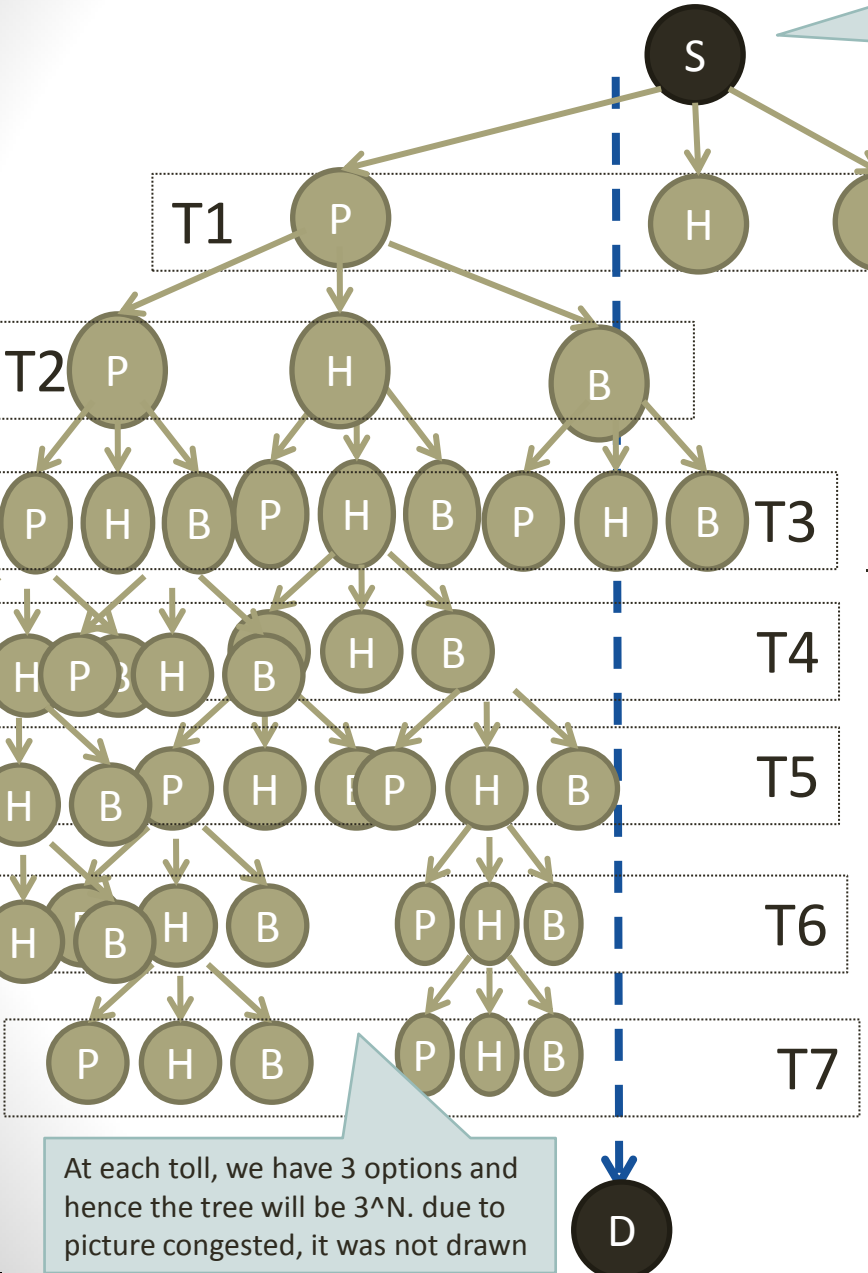
7 //toll no.of.tolls
10 100 //toll hire men and toll cost
70 5
80 15
20 60
50 90
30 80
10 10
Min cost: 150



Practice

TollGate – Approach

Approach: At each toll, we have 3 options. Hence we need to explore all 3 paths. **This can be done with traversal of tree in preorder or dfs.** Both Pay and Hire, can move forward as we traverse, but we can proceed battle traversal only if it succeeds.



- Time complexity for 1 test case is 3^N
- if N is 20, then time complexity is 34,867,84410

How to reduce the computing cycle?

- All battle will not happen as we may not have enough men with us to fight at all tolls.
- Also there is no point in traversing all the path if already computed cost is minimum than newly computed cost and hence it can be abandon mid way

T1 - Toll gate1

P Pay toll

H Hire and pay double toll

B Battle , do not pay toll

TollGate – Problem solving approach

```
int N,  t_cost  [22], t_hire[22], min_cost = 1000000;
```

```
void dfs (int tp, int cc)
{
if (tp == N-1) //base condition to check last toll gate
   {
          cc += tc[tp];
          if ( cc < min_cost) min_cost = cc;
          return;
   }

    dfs(tp+1, cc+tc[tp]); //toll pay option

}
```

Let us assume this is code that will help to traverse **toll pay** option only.

```
void dfs (int tp, int bp3, int bp2, int bp1, int cc)
{

   if (tp == N-1) //base condition to check last toll gate
   {
          cc += tc[tp];
          if ( cc < min_cost) min_cost = cc;
          return;
   }

   dfs(tp+1, bp3+th[tp], bp2, bp1 , cc+2*tc[tp]); //toll hire option

}
```

Similarly this will be the code that will help to traverse **toll hire and double toll pay option** only.

tp- toll position, **cc-** current cost, **tc-**toll cost, **th**-toll hire, **bp1-bp2-bp3** – battle pool

TollGate – Problem solving approach

```
int N,  t_cost  [22], t_hire[22], min_cost = 1000000;
void dfs(int tp, int bp3, int bp2, int bp1, int cc)
{
    int tot_bp = bp3 + bp2 + bp1;

    if (tp == N-1) //base condition to check last toll gate
    {
           if ( tot_bp < th[tp]) cc += tc[tp];
           if ( cc < min_cost) min_cost = cc;
           return;
    }


    if ( tot_bp >= th[tp] ) //toll battle option
    {
           if ( th[tp] > bp2 + bp1 )
           {
              bp3 = tot_bp - th[tp];
              bp1 = bp2 = 0;
           }
           else if ( th[tp] > bp1 )
           {
              bp2 = (bp1+bp2) - th[tp];
              bp1 = 0;
           }
           dfs(tp+1, 0, bp3 , bp2, cc); //note: pool3 is zero, pool3 becomes
pool2 and pool2 as pool1
    }
}
```

This piece of code for **toll battle** option only.

tp- toll position**, cc-** current cost**, tc-**toll cost, **th**-toll hire, **bp1-bp2-bp3** – battle pool

TollGate – Problem solving approach

```
int N,  t_cost  [22], t_hire[22], min_cost = 1000000;
  void dfs(int tp, int bp3, int bp2, int bp1, int cc)
  {
      int tot_bp = bp3 + bp2 + bp1;

      if (tp == N-1) //base condition to check last toll gate
      {
            if ( tot_bp < th[tp]) cc += tc[tp];
            if ( cc < min_cost) min_cost = cc;
            return;
      }

       dfs(tp+1, bp3 , bp2, bp1 , cc+tc[tp]); //toll pay option
       dfs(tp+1, bp3+th[tp], bp2, bp1 , cc+2*tc[tp]); //toll hire option

      if ( tot_bp >= th[tp] ) //toll battle option
      {
            if ( th[tp] > bp2 + bp1 )
            {
            bp3 = tot_bp - th[tp];
            bp1 = bp2 = 0;
            }
            else if ( th[tp] > bp1 )
            {
            bp2 = (bp1+bp2) - th[tp];
            bp1 = 0;
            }
            dfs(tp+1, 0, bp3 , bp2, cc); // note: pool3 is zero, pool3 becomes pool2
  and pool2 as pool1
      }
  }
```

tp- toll position, **cc-** current cost, **tc-**toll cost, **th**-toll hire, **bp1-bp2-bp3** – battle pool

Merge all 3 codes.
Is this efficient?
Time complexity for N=20 is 3^N

TollGate – Problem solving approach

```
int N,  t_cost  [22], t_hire[22], min_cost = 1000000;
  void dfs(int tp, int bp3, int bp2, int bp1, int cc)
  {
      int tot_bp = bp3 + bp2 + bp1;
      if (cc > min_cost) return; // condition important to avoid unnecessary cpu cycle
      if (tp == N-1) //base condition to check last toll gate
      {
            if ( tot_bp < th[tp] ) cc += tc[tp];
            if ( cc < min_cost) min_cost = cc;
            return;
      }

       dfs(tp+1, bp3 , bp2, bp1 , cc+tc[tp]); //toll pay option
       dfs(tp+1, bp3+th[tp], bp2, bp1 , cc+2*tc[tp]); //toll hire option

      if ( tot_bp >= th[tp] ) //toll battle option
      {
            if (th[tp] > bp2 + bp1 )
            {
            bp3 = tot_bp - th[tp];
            bp1 = bp2 = 0;
            }
            else if (th[tp] > bp1 )
            {
            bp2 = (bp1+bp2) - th[tp];
            bp1 = 0;
            }
            dfs(tp+1, 0, bp3 , bp2, cc); // note: pool3 is zero, pool3 becomes pool2
  and pool2 as pool1
      }
  }
```

This condition will avoid **unnecessary traversal** if the cost is going more than already computed min cost.

Rotating Battle pool members 3 to 2 and 2 to 1 pool

tp- toll position, **cc-** current cost, **tc-**toll cost, **th**-toll hire, **bp1-bp2-bp3** – battle pool

TollGate – Main function

```
#include <stdio.h>
#include<time.h>
// no.of.toll gate(between 5 and 20, cost at toll gate, total hire available at tollgate,
minimum cost
int N, tc [22], th[22], min_cost = 1000000;
void dfs(int tp, int bp3, int bp2, int bp1, int cc);
int main()
{
      int i, TC;
      clock_t start, end;
      double cpu_time_used;

      printf("No.of.TC? "); scanf("%d", &TC);
      start = clock();
      while( TC-- )
      {
            scanf("%d", &N);
            for ( i = 0; i < N; ++i)
             scanf("%d %d", &th[i], & tc [i]);


            dfs(0, 0, 0, 0, 0);
            printf("\nMinCost= %d\n\n", min_cost );
            min_cost = 1000000; //some large number
      }
      end = clock();
      cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
      printf("fun() took %f seconds to execute \n", cpu_time_used);
      getch();
     return 0;
}
```

Mar_Adv_TollGate-sasi-v3.c.txt

TollGate_input.txt

TollGate-Sasi(AdvSWMar16).7z

tp- toll position**, cc-** current cost**, tc-**toll cost, **th**-toll hire, **bp1-bp2-bp3** – battle pool

TollGate – Output

```
No.of.TC? 5
7
10 100
70 5
80 15
20 60
50 90
30 80
10 10

MinCost= 150


9
600 800
300 400
300 400
1000 400
300 600
100 300
600 300
600 500
1000 300

MinCost= 3000


11
1000 10
700 900
400 500
300 10
900 900
300 10
50 900
50 900
700 900
500 900
50 10

MinCost= 2370
```

```
20
896 546
543 216
454 310
408 367
40 602
252 582
954 627
850 234
763 479
232 278
301 538
528 508
936 154
629 443
758 336
432 700
882 256
278 738
517 882
317 136

MinCost= 4721


20
410 610
831 909
675 629
421 774
386 869
544 219
492 414
996 557
499 482
231 285
804 978
304 881
489 911
75 315
927 648
252 914
330 396
937 133
495 882
813 717

MinCost= 8231

fun() took 0.062000 seconds to execute
```

TollGate – cpp

```cpp
#include <iostream>
int N, cc[25], t[25], min_cost = 10000007;

void dfs(int p, int a, int b, int c, int cost)
{
	int asum = a+b+c;
	if (cost > min_cost) return;
	if (p == N-1)
	{
		if (asum < t[p]) cost += cc[p];
		if (cost < min_cost) min_cost = cost;
		return;
	}
	dfs(p+1, a, b, c, cost+cc[p]);
	dfs(p+1, a+t[p], b, c, cost+2*cc[p]);
	if (asum >= t[p])
	{
		if (t[p] > b+c) a = asum-t[p];
		if (t[p] > c) b = t[p]-c>=b ? 0 : b-t[p]+c;
		dfs(p+1, 0, a, b, cost);
	}
}
int main()
{
	std::cin >> N;
	for (int i = 0; i < N; ++i)
		std::cin >> t[i] >> cc[i];
	dfs(0, 0, 0, 0, 0);
	std::cout << min_cost << std::endl;
	return 0;
}
```

Thank You

SW Competency Advanced Exam

Tunnel Construction

28th Sept'16

Problem Statement

* There are V number of tunnels in parallel. A tunnel is a combination of blocks in horizontal direction. Number of blocks is given as H. So total tunnels will be V and total blocks is V*H.
* There are two construction machines, one at each end of the tunnel under construction.
* Each machine will work for a day and will construct one block of the tunnel but only one of the machines will work for a day, both can't work on the same day.
* A cost is associated with each machine, for working for one day.(C1 and C2 for machine 1 and 2 respectively)
* For every block there is a factor given as S, which will be multiplied by the cost of one day's work of a machine. So the cost of constructing one block will be S*C1 or S*C2.
* Additional cost is there if a machine is working for 2 or more consecutive days and is given as R1 and R2.
* Once we get to know the construction cost associated with all the tunnels(V) , we have to select N out of them such that the cost is minimum keeping in mind that there should be at least one tunnel between the chosen tunnels. Distance must at least be 2.
* A cost is associated with the movement of machines(M1 and M2) based on the distance between the construction sites(Tunnel distance) and is given as (M1*M1 + M2*M2)*D. D is the distance between chosen tunnels.

Output: The minimum cost associated with this construction of N Tunnels.

Inputs:
T as number of test cases, followed by test cases.
Each test case consists of:
N,H,V (in the first line)
V lines follow the first line, with factor S associated with each Block in one horizontal line. (H entries in each of the V lines).
C1,R1,M1(for machine 1)
C2,R2,M2(for machine 2)

Constraints:
1<=N<=5, 3<=H<=500, (N*2-1)<=V<=15, 1<=S<=300, 1<=C<=200, 1<=R<=500, 1<=M<=300

Approach

- Calculate normal construction cost for every possible case by both the machines. By going through every possible case. starting with 0 working days for machine 1 and H for machine 2. Then keep on incrementing number of days for machine 1 and reduce for machine 2. Each time calculate normal cost.
- For every possible case, add the additional cost for a machine working on consecutive days.
- Calculating normal cost is easy, additional cost should be such that the consecutive days for the machines should be minimum. Can be achieved by making them work on alternate days for maximum times.

Note:- The example they gave for minimum cost as:
2->1->2->2->2 means that machine 1 will work on second day and machine 2 will work on 1st day, 3rd day, 4th day and the 5th day.
This does not mean that machine 1 will construct the second block and machine 2 will construct rest of the blocks.
The machines will construct the blocks in sequence, one machine from one end and the other machine from the other end. They will never cross each other.
This was the most important observation.

So according to given example, minimum additional cost will come for 2 consecutive days.
Ie. Machine 2 working on 3rd 4th and 5th day consecutively.
So we will add the cost for 2 days(4th and 5th).

One more possible case to get minimum additional cost could be
2->2->1->2->2 or 2->2->2->1->2. All will give same additional cost as the number of consecutive days for machine 2 is same. Each will give additional cost of 2 days.
We can analyze other cases also and can come to a general equation to get the minimum cost.
(D2-D1-1)*R2 or (D1-D2-1)*R1, depending upon which machine has worked for more days.
D1=> days machine 1 has worked, D2=> days machine 2 has worked.
Adding the above calculated cost to the normal cost and checking the minimum gives us the minimum cost to construct one tunnel.
Same way, calculate for all the possible tunnels and finally select N out of them.

Complexity

Requirement: Construct N tunnels
Max length of a tunnel is 500.
To calculate normal cost(S*C), for all the scenarios, total cases = 500*500.
A machine can construct all the bocks, or one machine can construct one block and other can construct the rest, or 2 blocks by machine 1 and rest by machine 2 and so on……

1) So total number cases for one tunnel = 500*500.
For Maximum of 15 Tunnels= 15*500*500.

2) Now to select N which can at max be 5, we have to calculate every possible case. Gives us the worst case complexity less than 7^5. (seven possible places for single tunnel)

3) So total complexity can at worst be < 15*500*500 + 7^5.


Solution_C


Input


Output