Problem Debrief

CSESoc x CPMSoc Programming Contest 2021

Isaiah Iliffe, Joseph Luo, Angus Ritossa

Problems

1. CSESoc and CPMSoc
2. Fruitful Purchase
3. Two Scoops
4. Typing Champ
5. Farmer Joe
6. Dodgy Dominoes
7. Ingredients Label
8. Connect Four Cheater
9. Gradient
10. Strange Store
11. Cylinder Climb
12. Train Network

1. CSESoc and CPMSoc

Problem

Given a string, determine whether it be rearranged to spell "CSESOC" and/or "CPMSOC".

1. CSESoc and CPMSoc

Problem

Given a string, determine whether it be rearranged to spell "CSESOC" and/or "CPMSOC".

Solution

1. CSESoc and CPMSoc

Problem

Given a string, determine whether it be rearranged to spell "CSESOC" and/or "CPMSOC".

Solution

- We simply want to check if each character of the string input has any character in "CSESOC" and/or "CPMSOC".

1. CSESoc and CPMSoc

Problem

Given a string, determine whether it be rearranged to spell "CSESOC" and/or "CPMSOC".

Solution

- We simply want to check if each character of the string input has any character in "CSESOC" and/or "CPMSOC".
- The only characters that are not in both "CSESOC" and "CPMSOC" are ’E’ and ’M’, respectively.

CSESOC and CPMSOC - Code

```
keys = input().strip()
csesoc = all(c in keys for c in 'CSESOC')
cpmsoc = all(c in keys for c in 'CPMSOC')
if csesoc and cpmsoc:
    print('BOTH')
elif cpmsoc:
    print('CPMSOC')
elif csesoc:
    print('CSESOC')
else:
    print('NEITHER')
```

2. Fruitful Purchase

Problem

The shop lets you buy a single apple for X dollars, a single banana for Y dollars, or an apple and a banana together for Z dollars. You want to buy exactly A apples and B bananas. What is the smallest amount of money you can spend?

2. Fruitful Purchase

Problem

The shop lets you buy a single apple for X dollars, a single banana for Y dollars, or an apple and a banana together for Z dollars. You want to buy exactly A apples and B bananas. What is the smallest amount of money you can spend?

Solution

2. Fruitful Purchase

Problem

The shop lets you buy a single apple for X dollars, a single banana for Y dollars, or an apple and a banana together for Z dollars. You want to buy exactly A apples and B bananas. What is the smallest amount of money you can spend?

Solution

- If the total price of an apple and a banana is more than the deal, we want to take the deal as many as possible (until we have the amount of apples or banana we want)

2. Fruitful Purchase - Code

```cpp
// Solution by Angus
// Naive O(a+b) solution
#include <bits/stdc++.h>
using namespace std;
int main() {
        int x, y, z, a, b;
        scanf("%d%d%d%d%d", &x, &y, &z, &a, &b);
        int ans = 0;
        while (a || b) {
                if (a && b && z < x+y) {
                        // Take the deal
                        a--;
                        b--;
                        ans += z;
                } else if (a) {
                        // Buy a
                        a--;
                        ans += x;
                } else {
                        // Buy b
                        b--;
                        ans += y;
                }
        }
        printf("%d\n", ans);
}
```

3. Two Scoops

Problem

There are N flavours of ice cream, each with a tastiness value. You want to pick the tastiest 2-scoop ice cream (with different flavours), however some flavours must go at the top, some on the bottom, and some can go on either the top or bottom.

3. Two Scoops

Problem

There are N flavours of ice cream, each with a tastiness value. You want to pick the tastiest 2-scoop ice cream (with different flavours), however some flavours must go at the top, some on the bottom, and some can go on either the top or bottom.

Solution

3. Two Scoops

Problem

There are N flavours of ice cream, each with a tastiness value. You want to pick the tastiest 2-scoop ice cream (with different flavours), however some flavours must go at the top, some on the bottom, and some can go on either the top or bottom.

Solution

- Find the tastiest top only flavour, the tastiest bottom only flavour and the tastiest and second tastiest either flavour.

3. Two Scoops

Problem

There are N flavours of ice cream, each with a tastiness value. You want to pick the tastiest 2-scoop ice cream (with different flavours), however some flavours must go at the top, some on the bottom, and some can go on either the top or bottom.

Solution

- Find the tastiest top only flavour, the tastiest bottom only flavour and the tastiest and second tastiest either flavour.
- Consider the options: top + bottom, either + bottom, top + either, either + either

4. Typing Champ

Problem

There are N flavours of ice cream, each with a tastiness value. You want to pick the tastiest 2-scoop ice cream (with different flavours), however some flavours must go at the top, some on the bottom, and some can go on either the top or bottom.

4. Typing Champ

Problem

There are N flavours of ice cream, each with a tastiness value. You want to pick the tastiest 2-scoop ice cream (with different flavours), however some flavours must go at the top, some on the bottom, and some can go on either the top or bottom.

Solution

4. Typing Champ

Problem

There are N flavours of ice cream, each with a tastiness value. You want to pick the tastiest 2-scoop ice cream (with different flavours), however some flavours must go at the top, some on the bottom, and some can go on either the top or bottom.

Solution

- Intuition: initially set the number = 1. If the commands are 'U' or 'D', you want to decrement or increment the number by 3, respectively. Similarly with 'L' or 'R', decrement or increment the number by 1.

4. Typing Champ

Problem

There are N flavours of ice cream, each with a tastiness value. You want to pick the tastiest 2-scoop ice cream (with different flavours), however some flavours must go at the top, some on the bottom, and some can go on either the top or bottom.

Solution

- Intuition: initially set the number = 1. If the commands are 'U' or 'D', you want to decrement or increment the number by 3, respectively. Similarly with 'L' or 'R', decrement or increment the number by 1.
- Once we got 'P', we will store the number into a passcode array.

5. Farmer Joe

Problem

Given a square grid of integers. For each cell, its score is the sum of all cells NOT in a line N, S, E, W, NE, NW, SE or SW from it. Find the maximum score of any cell.

5	4	2	5	3
4	3	1	3	2
2	1	2	2	5
1	2	1	4	1
3	3	4	1	5

5. Farmer Joe

Problem

Given a square grid of integers. For each cell, its score is the sum of all cells NOT in a line N, S, E, W, NE, NW, SE or SW from it. Find the maximum score of any cell.

Solution

5. Farmer Joe

Problem

Given a square grid of integers. For each cell, its score is the sum of all cells NOT in a line N, S, E, W, NE, NW, SE or SW from it. Find the maximum score of any cell.

Solution

- Try every point as the centre, and print the maximum score.

5. Farmer Joe

Problem

Given a square grid of integers. For each cell, its score is the sum of all cells NOT in a line N, S, E, W, NE, NW, SE or SW from it. Find the maximum score of any cell.

Solution

- Try every point as the centre, and print the maximum score.
- For each centre, we can find the sum when it is the centre in $O(N^2)$ (overall solution $O(N^4)$).

5. Farmer Joe

Problem

Given a square grid of integers. For each cell, its score is the sum of all cells NOT in a line N, S, E, W, NE, NW, SE or SW from it. Find the maximum score of any cell.

Solution

- Try every point as the centre, and print the maximum score.
- For each centre, we can find the sum when it is the centre in $O(N^2)$ (overall solution $O(N^4)$).
- Idea: find the sum of the cells not included, and subtract from the total. Can find the sum of the cells not included in $O(N^3)$ (overall solution $O(N^3)$).

5. Farmer Joe

Problem

Given a square grid of integers. For each cell, its score is the sum of all cells NOT in a line N, S, E, W, NE, NW, SE or SW from it. Find the maximum score of any cell.

Solution

- Try every point as the centre, and print the maximum score.
- For each centre, we can find the sum when it is the centre in $O(N^2)$ (overall solution $O(N^4)$).
- Idea: find the sum of the cells not included, and subtract from the total. Can find the sum of the cells not included in $O(N^3)$ (overall solution $O(N^3)$).
- Better idea: pre-compute the sum of each row, column and diagonal. Then we can find the sum of cells not included in $O(1)$ (overall solution $O(N^2)$).

5. Farmer Joe - Code

```
#include <stdio.h>
#define MAXN 2010
int N, A[MAXN][MAXN], total, row[MAXN], column[MAXN], diag1[MAXN], diag2[MAXN], ans;
int main() {
  scanf("%d", &N);
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      scanf("%d", &A[i][j]);
      total += A[i][j];
      diag1[i+j] += A[i][j];
      diag2[i-j+N] += A[i][j];
      row[i] += A[i][j];
      column[j] += A[i][j];
    }
  }
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      int am = total - diag1[i+j] - diag2[i-j+N] - row[i] - column[j] + A[i][j]*3;
      if (am > ans) {
        ans = am;
      }
    }
  }
  printf("%d\n", ans);
}
```

6. Dodgy Dominoes

Problem

Given a sequence of dominoes in a line, each with a weight, determine the minimum number of pushes to knock them all over. Dominoes can only knock over dominoes of lower or equal weight.

Sample Input

```
6
2 2 3 2 1 10
```

Sample Output

```
3
```

6. Dodgy Dominoes

Problem

Given a sequence of dominoes in a line, each with a weight, determine the minimum number of pushes to knock them all over. Dominoes can only knock over dominoes of lower or equal weight.

Solution

- Can reframe the problem to decomposing the sequence into as few weakly monotonic sequences as possible.
- Go from left to right, keeping track of whether you are in an increasing sequence, decreasing sequence, or either (in the case of a streak of things of the same weight).
- Alternatively, after collapsing streaks of the same weight, the answer is something like $1+$ (number of local maxes) $+$ (number of local mins) $-$ (number of pairs of adjacent local maxes and mins).

6. Dodgy Dominoes - Code

```c
#include <stdio.h>

int N, w[100005], ans = 1;
int mode; // 0: undecided, 1: increasing, 2: decreasing

int main() {
    scanf("%d", &N);
    scanf("%d", &w[0]);
    for (int i = 1; i < N; i++) {
        scanf("%d", &w[i]);
        if (w[i] > w[i-1]) {
            if (mode == 0) mode = 1;
            if (mode == 2) mode = 0, ans++;
        }
        if (w[i] < w[i-1]) {
            if (mode == 0) mode = 2;
            if (mode == 1) mode = 0, ans++;
        }
    }
    printf("%d\n", ans);
}
```

7. Ingredients Label

Problem

There is a food which has N ingredients listed on the back. Originally, this was accompanied by the correct percentages of each ingredient, $p_1, p_2, ..., p_N$. It was known that

1. *p is non-increasing*
2. *each p_i is a positive integer*
3. *sum of all p_i is 100.*

But the company sneakily deleted some percentages. Find a possible allocation or say none exists

Sample Input: 50 -1 -1 10 -1
Sample Output: 50 20 10 10 10 (there are other possible correct answers, such as 50 20 19 10 1)

7. Ingredients Label

Problem

There is a food which has N ingredients listed on the back. Originally, this was accompanied by the correct percentages of each ingredient, $p_1, p_2, ..., p_N$. It was known that

1. *p is non-increasing*
2. *each p_i is a positive integer*
3. *sum of all p_i is 100.*

But the company sneakily deleted some percentages. Find a possible allocation or say none exists

- Each unknown value has a maximum and minimum based on the values to its left and right

7. Ingredients Label

Problem

There is a food which has N ingredients listed on the back. Originally, this was accompanied by the correct percentages of each ingredient, $p_1, p_2, ..., p_N$. It was known that

1. *p is non-increasing*
2. *each p_i is a positive integer*
3. *sum of all p_i is 100.*

But the company sneakily deleted some percentages. Find a possible allocation or say none exists

- Each unknown value has a maximum and minimum based on the values to its left and right
- First assign every value its maximum possible value.

7. Ingredients Label

Problem

There is a food which has N ingredients listed on the back. Originally, this was accompanied by the correct percentages of each ingredient, $p_1, p_2, ..., p_N$. It was known that

1. *p is non-increasing*
2. *each p_i is a positive integer*
3. *sum of all p_i is 100.*

But the company sneakily deleted some percentages. Find a possible allocation or say none exists

- Each unknown value has a maximum and minimum based on the values to its left and right
- First assign every value its maximum possible value. If the sum is bigger than 100, loop from right to left and decrease the unknown values until the sum is 100.

7. Ingredients Label

Problem

There is a food which has N ingredients listed on the back. Originally, this was accompanied by the correct percentages of each ingredient, $p_1, p_2, ..., p_N$. It was known that

1. *p is non-increasing*
2. *each p_i is a positive integer*
3. *sum of all p_i is 100.*

But the company sneakily deleted some percentages. Find a possible allocation or say none exists

- Each unknown value has a maximum and minimum based on the values to its left and right
- First assign every value its maximum possible value. If the sum is bigger than 100, loop from right to left and decrease the unknown values until the sum is 100. If we can't do this, then it's impossible.

8. Connect Four Cheater

Problem

Given a Connect Four grid (6 rows and 7 columns, where each cell is red, blue or empty), output a sequence of moves by the red player of minimum length that gets four in a row.

Sample Input

```
.......
.......
...R...
...B...
..BRR..
.BRBRBR
```

Sample Output

```
5 5
```

8. Connect Four Cheater

Problem

Given a Connect Four grid (6 rows and 7 columns, where each cell is red, blue or empty), output a sequence of moves by the red player of minimum length that gets four in a row.

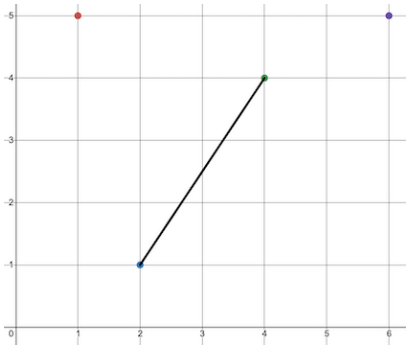
Solution

- Try every possible location of a four in a row.
  - Check if it is possible, that is, there are no blue things in the way.
  - Calculate minimum moves to get to it, that is, for each empty cell you want to fill, add one for the difference between original and final height (column is slightly different but easier).
- Alternatively, try all move sequences recursively (making moves in columns in non-decreasing order), or by trying every possible number of moves in each column. $O(6 \times 7 \times 6^7)$.

9. Gradient

Problem

Given a list of N points in 2D space, no two sharing an x-coordinate, find the largest gradient of any line passing through two points.
$N \leq 200\,000$.



9. Gradient

Problem

Given a list of N points in 2D space, no two sharing an x-coordinate, find the largest gradient of any line passing through two points.
$N \leq 200\,000$.

- Observation: Sort points by x-coordinates, the biggest slope will always be between two points which are adjacent.

9. Gradient

Problem

Given a list of N points in 2D space, no two sharing an x-coordinate, find the largest gradient of any line passing through two points.
$N \leq 200\,000$.

- Observation: Sort points by x-coordinates, the biggest slope will always be between two points which are adjacent.
- Solution: sort by x coordinate and compare adjacent points.

10. Strange Shop

Problem

Given an array P of integers, determine the largest absolute sum of any subarray which is less than or equal to M.

Sample Input

```
3 2
3 -4 7
```

Sample Output

```
1
```

10. Strange Shop

Problem

Given an array P of integers, determine the largest absolute sum of any subarray which is less than or equal to M.

Solution

- $O(N^3)$: calculate the sum of each range naively.

10. Strange Shop

Problem

Given an array P of integers, determine the largest absolute sum of any subarray which is less than or equal to M.

Solution

- $O(N^3)$: calculate the sum of each range naively.
- $O(N^2)$: calculate the sum of each range using prefix sums or similar.

10. Strange Shop

Problem

Given an array P of integers, determine the largest absolute sum of any subarray which is less than or equal to M.

Solution

- $O(N^3)$: calculate the sum of each range naively.
- $O(N^2)$: calculate the sum of each range using prefix sums or similar.
- $P[i] > 0$: for each left endpoint, calculate the rightmost valid endpoint that keeps sum under or equal to M, using two pointers or binary search or segment tree.

10. Strange Shop

Problem

Given an array P of integers, determine the largest absolute sum of any subarray which is less than or equal to M.

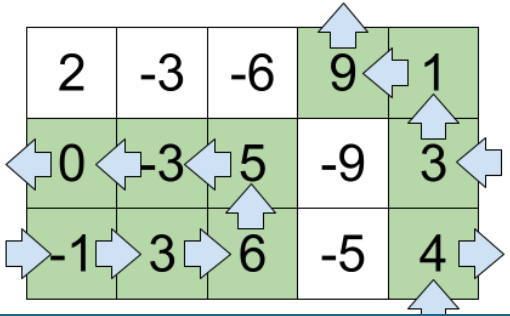
Solution

- $O(N^3)$: calculate the sum of each range naively.
- $O(N^2)$: calculate the sum of each range using prefix sums or similar.
- $P[i] > 0$: for each left endpoint, calculate the rightmost valid endpoint that keeps sum under or equal to M, using two pointers or binary search or segment tree.
- $O(N \log N)$: calculate prefix array pre, and determine largest $pre[r] - pre[l-1]$ which is less than or equal to M, using two pointers on sorted array. Allowing $r < l$ accounts for absolute value for free.

11. Cylinder

Problem

Given an $R \times C$ grid with point values, find the maximum walk from some square on the bottom row to some square on the top row. You can only move up, left or right, and can wrap around the grid between column 0 and $C - 1$. You cannot step on a square more than once. $R, C \leq 1\,000$.



11. Cylinder

Problem

Given an $R \times C$ grid with point values, find the maximum walk from some square on the bottom row to some square on the top row. You can only move up, left or right, and can wrap around the grid between column 0 and $C - 1$. You cannot step on a square more than once. $R, C \leq 1\,000$.

- We will use dynamic programming.

11. Cylinder

Problem

Given an $R \times C$ grid with point values, find the maximum walk from some square on the bottom row to some square on the top row. You can only move up, left or right, and can wrap around the grid between column 0 and $C - 1$. You cannot step on a square more than once. $R, C \leq 1\,000$.

- We will use dynamic programming.
- $O(RC^2)$ dp: state is the row and column, and recurrence is you try every other cell on the row as the first cell you visited in the row.

11. Cylinder

Problem

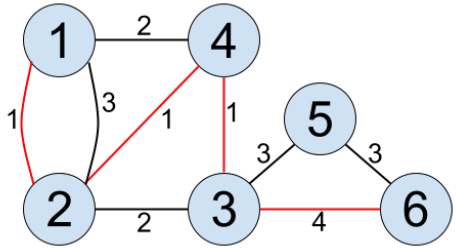
Given an $R \times C$ grid with point values, find the maximum walk from some square on the bottom row to some square on the top row. You can only move up, left or right, and can wrap around the grid between column 0 and $C - 1$. You cannot step on a square more than once. $R, C \leq 1\,000$.

- We will use dynamic programming.
- $O(RC^2)$ dp: state is the row and column, and recurrence is you try every other cell on the row as the first cell you visited in the row.
- $O(RC)$ use prefix sum to do the recurrence efficiently.

12. Train Network

Problem

Given an undirected weighted graph, find the cheapest path from node 1 *to node* N*, where a path's cost is the sum of the two largest edge weights on it. There is no edge between node* 1 *and node* N*, and* $N, M \leq 100\,000$*.*



12. Train Network

Problem

Given an undirected weighted graph, find the cheapest path from node 1 to node N, where a path's cost is the sum of the two largest edge weights on it. There is no edge between node 1 and node N, and $N, M \leq 100\,000$.

Solution

- $O(N^3)$: for every pair of edges, see if there exists a valid path using only edges cheaper than both edges in that pair, using DFS or similar.

12. Train Network

Problem

Given an undirected weighted graph, find the cheapest path from node 1 to node N, where a path's cost is the sum of the two largest edge weights on it. There is no edge between node 1 and node N, and $N, M \leq 100\,000$.

Solution

- $O(N^3)$: for every pair of edges, see if there exists a valid path using only edges cheaper than both edges in that pair, using DFS or similar.
- $\tilde{O}(N^2)$: try every edge as the most expensive, and find the minimum maximum edge cost to then connect node 1 and node N, using Prim's, or MST, or binary search.

12. Train Network

Problem

Given an undirected weighted graph, find the cheapest path from node 1 to node N, where a path's cost is the sum of the two largest edge weights on it. There is no edge between node 1 and node N, and $N, M \leq 100\,000$.

Solution

- $O(N^3)$: for every pair of edges, see if there exists a valid path using only edges cheaper than both edges in that pair, using DFS or similar.
- $\tilde{O}(N^2)$: try every edge as the most expensive, and find the minimum maximum edge cost to then connect node 1 and node N, using Prim's, or MST, or binary search.
- $\tilde{O}(N)$: precalculate minimum maximum edge cost to each node from node 1 and from node N, then try every edge as the most expensive. Edge can only be a most expensive edge if it's more expensive then minimum maximum both ways.