Solutions to the problems

By the Problem Set Committee

Algorithms Honors (prob1)

- Note that the valid grades are A, B, and C (inclusive).
- Also note that any GPA that is greater or equal to 2.50 is valid.
- If both grade and GPA are valid then a student is Eligible, otherwise he/she is not.
**Trees (prob2)**

- Length of the string is up to 500,000, so an efficient algorithm is required.
- Any prefix of a correct parenthesization will have a greater or equal number of "<" compared to ">". This can be used to distinguish left and right subtrees of a current tree in a string.
- Recursively parse the string and build a tree.
- During or after construction check if the maximum element in the left subtree is less than the value of the current node – BST condition.
- Also, check if the minimum element in the right subtree is greater than the value of the current node – BST condition.
- Finally check the heights difference (should be at most 1) – AVL condition.
- Divide & conquer algorithm will work in O(n logn) time and O(n) memory which is good enough for the input size.
- Note: Check for empty trees.

**Bull’s eye (prob3)**

- In this problem you need to find first the distance from each dart to the origin.
- Then, based on this distance, you should iterate through the concentric circles (or annulus) and find smallest one that contains it, and add the appropriate score. If the dart lands outside the board, you should subtract 5.
- This solution will run in O(n^2) which is good enough for the input size.
- However, one can improve it by searching efficiently using binary search to O (n*logn).
Paintball (prob4)

- Assume we know a minimum/maximum value for some subset of students. We use this information to find the minimum/maximum value for the bigger subset by adding a new pair of students.
- DP for minimum/maximum value based on all possible subsets of students. Overall around $2^n$ subsets will be considered – \( O(2^n \cdot n^2) \) time and \( O(2^n) \) memory.
- Start with \( \text{min}[0] = 0 \), and \( \text{min}[i] = \infty \) for \( i > 0 \)
- Then for every subset \( i \), where \( \text{min}[i] \) is defined, try to add a pair of students (\( A \) and \( B \)) such that neither \( A \) nor \( B \) are in the subset \( i \), then \( \text{min}[i + \text{team competitiveness of (students A and B)}] \) will be \( \text{min}[i] + \text{team competitiveness of (students A and B)} \)
- Alternatively, one could use memorization based on the same recurrence relation.
- The problem can be solved in polynomial time using Minimum Cost Maximum Flow, but was not required given the input size.

Marbles! (prob5)

- Problem asks you to compute number of triples of points that lie on a line, i.e. if there are four points that all lie on a line then, there are four “triples” of points.
- A naive solution in \( O(n^3) \) – loop through all distinct triple of points and increase counter if they all lie on a same line.
- To check if three points lie on a line, consider the triangle formed by these points. The points lie on a single line if and only if the area of the triangle is zero
- Cross product gives a convenient way to compute the area of the triangle

```cpp
for(int i=0;i<n;i++)
    for(int j=i+1;j<n;j++)
        for(int k=j+1;k<n;k++)
            if(crossProduct(points[i], points[j], points[k])==0) counter++;
```

- With a bit more of thinking, the problem can be solved in \( O(n^2 \log n) \) time:
  - Fix a point and translate it to the origin – \( O(n) \)
  - Sort all points by their angles – \( O(n \log n) \)
  - If \( k \) points have same angle, then it is possible to choose two points in \( O(k^2) \) ways and form a triple of points (the one on the origin, and any two chosen from \( k \) points) that lie on a line – \( O(n) \).
  - Overall \( O(n \log n) \) for a single point, and \( O(n^2 \log n) \) for all points.
Crazy-Rectangular Sea (prob6)

- Although the constraints for \( N \) and \( M \) are small, BFS solution will run out of time; since the depth of the BFS tree will be up to 5000, i.e. \( \approx 4^{5000} \) of nodes.
- Model the problem as a Graph Theory problem:
  - Every cell in the Sea is a single vertex of the graph
  - An edge between vertexes \( u \) and \( v \) exists if and only if it is possible to move from \( u \) to \( v \) in a single step.
  - The weight of the edge is the probability of the move (i.e. 0.5, 0.33, or 0.25)

**Note:** The Destination cell has no outbound edges.

- Notice that if adjacency matrix of the graph is squared (matrix multiplication), then the edge \((u, v)\) in the resulting matrix will represent the sum of weights of all paths of length two from \( i \) to \( v \). The weight of a path is the probability that ship will follow the path.
- The above property folds true for any \( k \)-th power of an adjacency matrix for any non-negative integer \( k \).
- Using this property one can iteratively multiply the adjacency matrix and get the weights for all paths from Start to Destination of all possible length, and sum them up. This will take \( O(L^3 \cdot P) \), where \( L = N \cdot M \) and \( P \) is the Magic-Pirate-Number which is not good enough.
- Notice that if the Destination cell had a self loop of weight 1, then the powering of the adjacency matrix would automatically sum the weights of all possible paths; so using fast exponentiation for the powering of the matrix, we produce a faster solution that runs in time \( O(L^3 \cdot \log P) \) which is acceptable.

The Maximum-Minimum Theorem (prob7)

- Test the function at both ends \( a \) and \( b \)
- Test the function in the range \([a, b]\) with increment \( t = (b - a) / s \)
  - **Note:** floating point issues could occur if you use statements such as: 
    for\((x=a;x<=b;x+=t)\), instead it should be for\((x=a;x<=b+EPSILON;x+=t)\)
  - From all the evaluations choose the minimum/maximum values and store the \( x \) positions along with the minimum/maximum evaluations
  - Important: do NOT perform any rounding/truncating during the entire process
  - Check the output format: spacing, punctuations, etc
Hidden Squares (prob8)

- You are given up to 13 digits, and you want to find all possible numbers \(X\) that you can construct using these 13 digits only, such that the square of \(X\) is also constructible with these 13 digits.
- Assume that \(N\) is the maximum number that one can construct with the given digits (i.e. sort the digits in non-increasing order). Notice that \(X \leq \sqrt{N}\), since \(X^2 \leq N\).
- Using this observation it is OK to iterate through all integers from 1 to \(\sqrt{N}\) and check if integer \(i\) and \(i^2\) are constructible.

- Alternative solution:
  - Pick any subset of digits from given 13 digits. Take all possible permutations of the subset – this will produce all the numbers that one can construct with given subset of digits. Check if any of those numbers are “Hidden Pairs”.

Optimal Grading (prob9)

- Observe that maximizing \(- (p_1 \log p_1 + p_2 \log p_2 + \cdots + p_n \log p_n)\) is equivalent to minimizing \(p_1 \omega_1 : p_2 \omega_2 : p_3 \omega_3 : \cdots : p_n \omega_n\).
- Observe also that order of the grades’ percentages does not change, i.e. the order is independent of the placements of the cutoffs.
- DP on two parameters \(G\) and \(C\). \(G\) is number of different grades left to consider, \(C\) is number of cutoffs left to select.
- At any position you can choose to set a cutoff at current position or to skip it.
- When you are out of cutoffs, count how many students you have in every grade scale, and compute the answer.
Odd Matrices (prob10)

- It is possible to fill the entire matrix except the last row and the last column in $2^{(n-1)(m-1)}$ different ways.
- All the elements of the last column and all the elements of the last row are forced to be 0 or 1 based on the filling of (N-1) x (M-1) submatrix (Except the one in the right bottom corner)
- It is possible to choose 0 or 1 for the element in the right bottom corner (intersection of the last column and the last row) if and only if parities of N and M are same. Otherwise it would have to be 0/1 for last column and 1/0 for the last row, which produces contradiction.

*BONUS*: Online training resources:

- topcoder.com/tc:
  - Topcoder is an online programming competition website where you can compete individually against other contestants (usually about 3 times a month). It also contains algorithm tutorials.
- spoj.pl, acm.uva.es, acm.timus.ru:
  - These sites are online judge systems that contain ACM ICPC programming competition problems.
- train.usaco.org/usacogate:
  - This is where the US High School Olympiad teams practice. It contains excellent algorithm tutorials, and each problems are accompanied with an explanation and solution.
- train.usaco.org/usacogate:
  - This is where the US High School Olympiad teams practice. It contains excellent algorithm tutorials, and each problems are accompanied with an explanation and solution.
- projecteuler.net:
  - This site contains problems inspired by the prolific Mathematician Euler. They are mathematical and algorithmic in nature.