Solutions

Leidsch Kampioenschap Programmeren

Preliminaries of the
Benelux Algorithm Programming Contest 2015

Universiteit Leiden

September 19, 2015

A general remark

- return 0

E: Board Game 3

- Read the input and add the position of each w to a queue

-
  ```
  while queue is not empty
  { take first element from queue
    for all 8 adjacent tiles
      if (empty)
        add new w tile to queue
  }
  ```

- Return number of new w's
- Note: w's do not have to be adjacent to each other (see sample data)

A: Board Game 1

Remember BAPC Leiden 2006

- Read input and store in array.
- Double tiles may be ignored.
- Apply a series of if else statements, checking whether there are
  - three or more tiles with the same number and *different* colors.
  - three or more *consecutive* tiles of the same color.

Common mistakes:

- Can not be done directly from input, so use an array.
- Initialize array at the right size.

G: Sir Jumpsalot

- $X^2 + Y^2 = Z^2$ with $Z = J * \sqrt{D}$
- $X^2 + Y^2 = J^2 * D$
- $J^2 = (X^2 + Y^2)/D$
- take square root of $(X^2 + Y^2)/D$ and round up
- do **not** calculate $\sqrt{X^2 + Y^2}/\sqrt{D}$
- special case: if $0 < X^2 + Y^2 < D$, then answer is 2

K: Road Trip

- Straightforward
- Given the $N \leq 1,000\,000$, the solution should obviously be $O(N)$ and not $O(N^2)$

```
int tank = 0, mintank = 0, mini = 1;
for(int i=1; i<=N; i++)
  tank += G - D
  if(tank < mintank) {
    mini = i;
    mintank = tank;
  }
if(tank < 0)
  cout << "IMPOSSIBLE" << endl;
else
  cout << mini << endl;
```

H: Storm Damage

- connected components
- nodes: blocks
- (undirected) edges: power lines
- determine connected components
- output number of components without power source

F: Anagram

- ad hoc
- possible if most frequent letter occurs at most once more than all others together
- repeat
  - append most frequent letter if it occurs exactly once more than all others together
  - otherwise, append 'smallest' avalaible letter different from previous letter
- with strings: do not use 'anagram + ch' to append letters (time limit)

C: Jewellery

- Easy if you know formula to compute area of polygon:

  ```
  area = 0;
  for each edge (x1,y1)-(x2,y2) of polygon (in order)
    area += x1*y2 - y1*x2  (cross product)
  area = |area| / 2;
  ```

  divide area by area of triangle
- Otherwise:
  - keep track of 'vertical' lines per row
  - count how many triangles between lines are inside shape

  This might be too slow

B: Video Game

- BFS
- node for each pair of positions Pac-Men
- edge if N,E,S,W takes Pac-Men from one pair to another

J: Board Game 4

- DP

-
  ```
  set P[i][0]=1.0
  set P[1][j]=0.0

  for (i=2;i<=M;i++)
    for (j=1;j<=N;j++)
    { compute P[i][j]
          from P[i-2][j], P[i-1][j-1], P[i][j-2]
          and from P[i-1][j], P[i][j-1]
          (taking lowest value, best for defender)
    }
  ```
- $O(N * M * D^2)$ is OK
- $O(N * M * D^3)$ is not OK
- Note: choice of defender for 1 or 2 dice does not only depend on values of attacker, but also on *i* and *j*

D: Board Game 2

- graph + DP (subset sum)
- nodes of graph: players
- players A and B certainly belong to same team,
  if they have **not** played match against each other
- in that case: (undirected) edge between A and B
- yields complement graph
- determine connected components: $1, 2, \dots, C$, with size[i]
- component 1 contains player 1

D: Board Game 2 (2)

- array Teams[C][N]: number of teams including player 1

-
  ```
  set Teams[1][j]=0
  Teams[1][size[1]] = 1

  for (i=2;i<=C;i++)
    for (j=1;j<=N;j++)
      Teams[i][j] =
        Teams[i-1][j] + Teams[i-1][j-size[i]]
        (modulo 10ˆ8)
  ```
- return Teams[C][N]
- backtrace array to find particular team

I: Advanced Modelling

Simulation + geometry
Simulate the shot, linear in F and D.

- Line-plane intersection (where does it hit the face - normals + dot product)
- Distance from origin on line (what face does it hit first)
- 2D point in convex polygon (does the shot hit the face: use sign of cross product and the next step)
- Project face + projected point onto 2D (GramSchmidt process - project + dot product)
- mirror vector in 3D (does the shot deflect - projection)