One-Day Assignment 3 –algorithmic solution

Coconut Splat

One-Day Assignment 3 – Pseudocode (English Description)

We use a custom class, “Hand” to represent a hand, or hands in the case of folder hands. It contains the player which it belongs to (stored as an int), as well as the hand type it represents (can be stored as an int, but for clarity’s sake we shall use a String instead)

We use a linked list of Hands when solving this problem, which we shall name as just “list”

One-Day Assignment 3 – Pseudocode (English Description)

Initialise the list with num_player instances of hands. All hands are of type “folded”, but the hands belong to different players, from player 1 at the front of the list, to player num_player at the back of the list.

Simulate the counting off of hands by moving the hand at the front of the list to the back of the list (num_syllables – 1) times. Afterwards, remove the hand at the front of the list and store it as a variable curr_hand.

Look at the type of hand represented by curr_hand.

One-Day Assignment 3 – Pseudocode (English Description)

If it is “folded”, then add two instances of hands of type “fist” to the front of the list, with the same player number as curr_hand.

If it is “fist”, then add a new hand of type “palm” to the back of the list, with the same player number as curr_hand.

If it is “palm”, then nothing needs to be added to the list.

Repeat this simulation as many times as necessary until there is only one hand left. Output the player whom that hand belongs to.

One-Day Assignment 4 –algorithmic solution

Conformity

One-Day Assignment 4 – Pseudocode (General)

The following applies to both the code-like and English description:

We use a HashMap which uses an arraylist of integers (or optionally, just a long) as the key, and an integer as the value. This is used to count the number of occurrences of a combination of courses.

One-Day Assignment 4 – Pseudocode (code-like)

```
for curr_student in students:
    create new array(list) of integers arr
    for course_no in curr_student:
        add course_no to arr
    sort arr
    if arr[1] is in hash:
        update entry (arr -> value) in hash to (arr -> value + 1)
    else:
        add entry (arr -> 1) to hash
```

1. Directly hashing an arraylist of integers works, but you may manually convert to a long first if you wish

One-Day Assignment 4 – Pseudocode (code-like)

```
initialise total = 0, max = 0

for entry[2] in hash:
    if entry.value > max:
        total = entry.value; max = entry.value
    else if entry.value == max:
        total = total + entry.value

output total
```

2. Recall that enhanced for-loops do not work directly on a HashMap

One-Day Assignment 4 – Pseudocode (English Description)

For each student, put the list of courses the student takes into an array(list). To avoid the issue of having different ordering of the same course numbers, sort the array so that the numbers are always in a consistent order (increasing order).

Check if this list of courses is present in hash. If it is, update the value mapped to it, to (value + 1).

Otherwise, add a new entry for this list of courses, with the value set to 1.

One-Day Assignment 4 – Pseudocode (English Description)

Once done, iterate through hash to determine which combinations are the most popular. Start off with two variables: total, and max. Both are initialised to 0.

If we encounter an entry which has a value > than max, set both total and max to be = the entry's value. Otherwise, check if the value is == max. If so, add the entry's value to total.

Finally, output total as the final answer.

One-Day Assignment 5 –algorithmic solution

Assigning Workstations

One-Day Assignment 5 – Pseudocode (English Description)

We use an array(list) of IntegerPairs (or equivalent), to store the arrival time of each researcher, as well as how long they will use a workstation for.

We use two min heaps of integers to keep track of workstations. One min heap is used to keep track of when a workstation will be free for use (changing from “in use” to “unused and unlocked”), while the other is used to keep track of when an unused workstation will relock.

We also use an integer to store the final answer, initialised to 0

One-Day Assignment 5 – Pseudocode (English Description)

Sort the array(list) of IntegerPairs, so we can order the researchers based on who arrives first.

For each researcher that arrives (in chronological order), we do the following:

1. Check the “in use” heap for any workstations that are no longer in use (value in heap <=[1] time researcher arrives). For each such element, remove it from this heap, and add (value + m) to the “unused and unlocked” heap.
2. Check the “unused and unlocked” heap for any workstations that have relocked (value in heap <[1] time researcher arrives). For each such element, remove it from this heap.
3. If any elements remain in the “unused and unlocked” heap, remove the smallest one, and increment the answer by 1.
4. Add (time researcher arrives + time workstation will be used) to the “in use” heap

1. These two signs are different; this is not a typo

One-Day Assignment 6 –algorithmic solution

Kattis’s Quest

One Day Assignment 6 – Algorithm

- When a new quest is added, add it to your data structure – (1)
- When a query occurs:
  - While true:
    - quest = get_suitable_quest(remaining_energy) – (2)
    - if quest is null, break
    - answer += quest.quest_gold_reward
    - remaining_energy -= quest.quest_energy_cost
    - Remove quest from data structure – (3)
  - Output answer (should use a **long** instead of int)

One Day Assignment 6 – Representation

- The issue of TreeSet not supporting duplicate elements is the main concern for this problem
- The following slides document several possible ways to handle the 3 different parts of the algorithm in the previous slide (indicated with a (1), (2), or (3))
- In the following slides, *energy* refers to the energy cost of the quest, *reward* refers to the gold reward, and *remaining_energy* refers to the amount of energy left

One Day Assignment 6 – Representation

- Option 1: Use TreeSet<IntegerTriple>
- IntegerTriple contains 3 values:
  - First value: energy cost of the quest
  - Second value: gold reward of the quest
  - Third value: special unique id (to ensure we can support duplicates)
- (1) – Add new IntegerTriple (*energy*, *reward*, id++)
- (2) – floor() of IntegerTriple (*remaining_energy*, INF, INF), where INF is a large integer
- (3) – Remove the IntegerTriple found in step (2)

One Day Assignment 6 – Representation

- Option 2: Use TreeMap<Integer, PriorityQueue<Integer>>
- The key Integer contains 1 value:
  - Energy cost of the quest
- The Integer value in PriorityQueue (max heap) contains 1 value:
  - Gold reward of the quest
- (continued in next slide)

One Day Assignment 6 – Representation

- (cont. from previous slide)
- (1) – If *energy* is not present in the TreeMap, add a new entry mapping *energy* -> new PQ.
  Regardless of the above, find *energy* in the TreeMap, and add *reward* to the associated PQ.
- (2) – floorEntry() of *remaining_energy*, and peek from the associated PQ
- (3) – Pop from the PQ in (2). Remove entry from the TreeMap if the PQ is empty

One-Day Assignment 7 –algorithmic solution

Weak Vertices

One Day Assignment 7 – English Description

Use an array to mark whether a vertex is part of a triangle or not. Initially, all of them are set to false

Try all possible permutations of 3 vertices and see if they form a triangle. This can be achieved by the following:

Use a total of 3 nested for-loops from 1 to n

- For each iteration, check that the 3 values(a, b, c) in the loops are distinct. If they are all distinct, then lookup the adjacency matrix for whether the edges (a, b), (a, c), and (b, c) exist
- If they do exist, then set a, b, and c in the array to true

At the end, output all vertices that are not part of a triangle (value of false in the array)

One-Day Assignment 8 –algorithmic solution

Islands

One Day Assignment 8 – General

We keep the graph as is (in grid form)

We use an additional (2D) boolean array (visited_arr) to keep track of whether a cell in the grid has been visited or not. Initially, all cells are marked as unvisited

This problem can be solved through either BFS or DFS; if BFS is chosen, we will also need a Queue<IntegerPair> to keep track of cells in the BFS queue. We use an IntegerPair here as we represent a cell by (cell_row, cell_col) in the queue, as opposed to just a single integer.

If using DFS, you can use an explicit stack similarly to BFS, or just use recursion directly.

One Day Assignment 8 – BFS (loop in main)

answer = 0

for curr_cell in grid[1]:
    if curr_cell is ‘L’ && curr_cell is unvisited:
        answer = answer + 1
        add curr_cell to queue
        mark curr_cell as visited in visited_arr
        BFS(queue, grid, visited_arr)

output answer

1. In actual implementation, you will likely need to use 2 for loops to do this, instead of 1

One Day Assignment 8 – BFS Algorithm

Body of BFS(queue, grid, visited_arr[2]):

while queue is not empty:
    dequeue cell_to_check from queue
    for each of the four directions (up, down, left, right) from cell_to_check:
        next_cell = cell in the direction from cell_to_check
        if next_cell is not out of bounds, and is a ‘C’ or ‘L’, and is unvisited:
            add next_cell to queue
            mark next_cell as visited

2. It may help to pass in the dimensions of the grid as additional parameters as well, as opposed to calling .length on the grid to get the dimensions

One Day Assignment 8 – DFS (loop in main)

answer = 0

for curr_cell in grid[1]:
    if curr_cell is ‘L’ && curr_cell is unvisited:
        answer = answer + 1
        DFS(curr_cell, grid, visited_arr)

output answer

1. In actual implementation, you will likely need to use 2 for loops to do this, instead of 1

One Day Assignment 8 – DFS Algorithm

Body of DFS(curr_cell, grid, visited_arr[2]):

mark curr_cell as visited in visited_arr

for each of the four directions (up, down, left, right) from curr_cell:
    next_cell = cell in the direction from curr_cell
    if next_cell is not out of bounds, and is a ‘C’ or ‘L’, and is unvisited:
        DFS(next_cell, grid, visited_arr)

2. It may help to pass in the dimensions of the grid as additional parameters as well, as opposed to calling .length on the grid to get the dimensions

One Day Assignment 8 – English Description

We iterate through every cell in the grid. When we encounter a cell that is an ‘L’, and has not been visited yet, we increase a variable called answer (initialised to 0) by 1, and begin graph traversal (either BFS or DFS) on it.

During graph traversal, when looking through the 4 different directions from a cell, we first check:

- If the new position is out of bounds of the grid
- If not, we then check if the new position contains a ‘C’ or an ‘L’
- If it does, we then check that the new position has not been marked as visited yet

Only when the new position passes all 3 checks do we continue graph traversal on that cell, and mark it as visited

When done, output answer as the final result

Looping over directions

When trying to loop over the “directions” (up, down, left, right), it can be helpful to define direction arrays:

```
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};
```

So now to loop over directions, we can loop over the indices 0,1,2,3 to get the x and y direction.

You can even extend the “directions” to include diagonals!

One-Day Assignment 9 –algorithmic solution

Lost Map

One Day Assignment 9 – General

Run MST.

Additional notes:

For Prim's:

Consider storing an IntegerTriple in your PQ (store w, u, v), so you can keep track of the exact edges in the MST, since the problem requires the edges themselves, not just the total cost. Converting the graph to an Adjacency list not strictly necessary.

For Kruskal's:

Conversion to an Edge List would be needed, but as a default Edge List stores u and v already, getting the edges themselves may be easier than modifying Prim's.

One Day Assignment 9 – General

For Prim's variant for dense graphs:

Lecture slides example already uses the version which stores the edges themselves, which can be modified to output the edges instead

One Day Assignment 9 – Prim’s

convert AdjMat to AdjList (optional)

set vertex 1 (or 0, depending on numbering) to taken; add neighbours to PQ of IntegerTriples

while PQ is not empty:

    dequeue edge from PQ

    if v of edge is not taken:

        output edge (u, v)

        set v to taken

        add neighbours of v that are not taken to the PQ

One Day Assignment 9 – Prim's Variant

convert AdjMat to AdjList (optional)

initialise IntegerPair array A (set everything to (inf, -1)), and boolean array B (set everything to false (already done implicitly))

set A[1] (or A[0], depending on numbering) to (0, -1)

One Day Assignment 9 – Prim's Variant

```
while true:
    scan through A to find the pair with the smallest first value, as well as its index
    if pair.first is inf, break
    if pair.second is not -1:
        output (index, pair.second)
    B[index] = true; A[index] = inf
    for all outgoing edges (v, w) from index:
        if (!B[v] and A[v].first > w):
            A[v] = (w, index)
```

One Day Assignment 9 – Kruskal's

convert to edge list; sort edge list in ascending order of weight

set up UFDS

for edge in edge list (left to right):
    if (u, v) are not in the same set:
        union sets containing u and v
        output edge (u, v)

One-Day Assignment 10 –algorithmic solution

Human Cannonball Run

One Day Assignment 10 – General

- For this problem, it can be split into 2 parts:
  - Graph modelling
  - Running SSSP
- In this graph, vertices represent all points of interest in the conference (the starting point a, the ending point b, and all cannon)
- Edges represent the time taken to reach point v from point u directly (ie. not using any other cannon along the way, except for possibly the cannon at point u, if one exists)

One Day Assignment 10 – Graph Modelling

```
for vertex1 in vertices:
    for vertex2 in vertices:
        dist = sqrt(pow(vertex1.x – vertex2.x, 2) + pow(vertex1.y – vertex2.y, 2))[1]
        time1 = dist / 5
        time2 = INF // some large number
        if vertex1 is cannon:
            time2 = 2 + (abs(dist – 50) / 5)
        // add edge (vertex1 -> vertex2) with weight (min(time1, time2)) to graph DS of your choice
```

1. Alternatively, Java provides a Math.hypot() method

One Day Assignment 10 – Bellman Ford

Initialise a distance array. Set everything except the starting point (a) to a large number

```
for i from 1 to num_vertices - 1:
    for edge in graph:
        if (dist[edge.u] + edge.w < dist[edge.v]):
            dist[edge.v] = dist[edge.u] + edge.w
return dist[b]
```

One Day Assignment 10 – M. Dijkstra’s

Initialise a distance array. Set everything except the starting point (a) to a large number

add (0, a) to a PQ of IntegerPairs

```
while PQ is not empty:
    dequeue (dist_estimate, vertex) from PQ
    if dist[vertex] == dist_estimate:
        for each neighbour of vertex:
            if (dist[vertex] + edge.weight < dist[neighbour]):
                dist[neighbour] = dist[vertex] + edge.weight
                add (dist[neighbour], neighbour) to PQ
return dist[b]
```

One Day Assignment 10 – English Description

- Given two points (x_A, y_A) and (x_B, y_B), the direct distance between the two of them is:
  - $\sqrt[2]{(x_A - x_B)^2 + (y_A - y_B)^2}$
- After calculating the distance, we need to convert the distance into time required to cover the distance. Pick the minimum of the following two as the weight of edge (u, v)
  1. $\frac{dist}{5}$
  2. $2 + \frac{|dist - 50|}{5}$ (only if vertex u represents a point with a cannon)
     - |x| means the absolute value (ie. if x is negative, change it to positive)

One Day Assignment 10 – English Description

- After modelling the graph as in the previous slide, just run any suitable SSSP algorithm from point a, and find the distance at point b.
- Note that $E = O(V^2)$ since this is a complete graph. The possible options are:
  - Bellman Ford: $O(V^3)$
  - Original/Modified Dijkstra's: $O(V^2 \log V)$ (no negative edges)