Computer Programming 1

2023/12/5 Andy Hung

Slide here!

Outline

- hw09/hw09+
- Struct
- DFS
- BFS
- Debug strategies

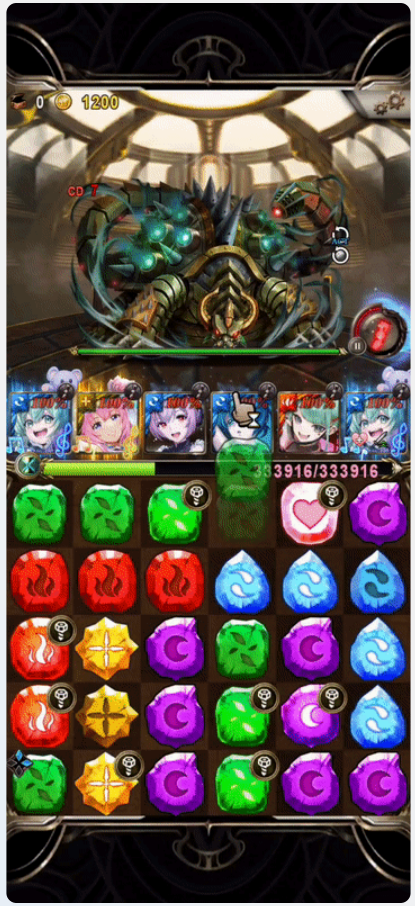
hw09

Tower of Saviors is one of a popular mobile game released in 2013. The game has been known for its special gameplay. In the game, the players will be given a stage with 5 rows and 6 columns and will be asked to dissolve the Runestones on the stage.

hw09

Runestones are tiles that players have to align horizontally or vertically with a minimum of 3 Runestones in order to dissolve them.

Aligning at least 3 Runestones of the same attribute will give the players' monsters power to attack or recovery, corresponding with the attributes of the Runestones, which will be Water, Fire, Earth, Light, and Dark and Health.



hw09

In this assignment, you are asked to calculate the final result of monsters' value of attack and recovery by dissolving the given stage.

Notice that in order to simplify the assignment, we do not need to count the combo. But you should dissolve the Runestones repeatedly until there is no any at least 3 Runestones with the same attribute in the stage.

hw09

Input

The input starts with six members in your team. Each team member will provide their **TITLE**, **NAME**, **TYPE**, **ATTACK**, and **RECOVERY**.

As in sample input, the beginning and the end of TITLE are indicated by "----". Other information follows the colon ":".

After the team members, there will be a line of twenty hyphens (--------------------). Please note that you may only provide the **LEADER** and **SUPPORTER**. The **MEMBER** can be None.

The next N lines consist of 6 characters each, ending with "------". These represent Runestones preparing to fall onto the stage. Your stage is under "------", and it consists of a 5x6 2D character array.

hw09

Input

There are 6 types of runestones. In the input, uppercase letters represent Reinforced Runestones, and lowercase letters represent Normal Runestones:

- Water: W w
- Fire: F f
- Earth: E e
- Light: L l
- Dark: D d
- Health: H h

Data Spec

- $N \leq 10000$
- $len(\text{Monster name}) < 100$
- $0 \leq \text{Monster attack} < 10000$
- $0 \leq \text{Monster recovery} < 10000$
- $\text{Monster type} \in \{\text{"Water"}, \text{"Fire"}, \text{"Earth"}, \text{"Light"}, \text{"Dark"}\}$

hw09

Output

You need to calculate the final Damage and Recovery for each member and then output the members from the highest Damage to the lowest Damage. You can use **qsort** to perform this operation.

Each member 's output should be separated by 43 hyphens "-". The name should be printed between two "|" characters, ensuring that there are precisely 41 characters between the two "|". The next line should display the Damage and Recovery, with 20 characters between each pair of "|".

hw09

Output

When calculating final damage and recovery, you can use the following formula:

> **Tips**
>
> $$\mathrm{RunestonesBase} = \mathrm{NormalRunestonesCount} \times 0.25 + \mathrm{ReinforcedRunestonesCount} \times 0.4 + \mathrm{RunestoneTypeComboCount} \times 0.25$$

Since this assignment is simplified, if runestones' count > 0, we assume one combo count.

hw09

Take this board as an example:

hw09

Take this board as an example:

- Some runestones dissolves:
  - 4 Earth runestones
  - 5 Fire runestones
  - 5 Water runestones
  - 3 Light runestones
  - 3 Dark runestones
  - 3 Earth runestones
  - 3 Dark runestones

hw09

Take this board as an example:

- Some runestones dissolves:
  - 4 Earth runestones
  - 5 Fire runestones
  - 5 Water runestones
  - 3 Light runestones
  - 3 Dark runestones
  - 3 Earth runestones
  - 3 Dark runestones
- Runestones fall down and fill the board.

hw09

Take this board as an example:

- Some runestones dissolves:
  - 4 Earth runestones
  - 5 Fire runestones
  - 5 Water runestones
  - 3 Light runestones
  - 3 Dark runestones
  - 3 Earth runestones
  - 3 Dark runestones
- Runestones fall down and fill the board.
- **Repeat** until no runestones can be dissolved.

hw09+

In the last assignment, you are asked to dissolve horizontally or vertically with a minimum of 3 Runestones. As the game ages, new types of dissolving strategies have been introduced.

In this assignment, you need to handle different **LEADER_SKILL**, which represents diffenent resolving strategy:

- THREE_ANY: Runestones can be dissolved by aligning 3 or more of them
- TWO_HEART_{TYPE}: Heart and **{TYPE}** Runestones can be dissolved by aligning 2 or more of them, which **{TYPE}** can be WATER, FIRE, EARTH, LIGHT, and DARK.

hw09+

Input

The input starts with six members in your team. Each team member will provide their **TITLE**, **NAME**, **TYPE**, **ATTACK**, **RECOVERY**, and LEADER_SKILL.

As in sample input, the beginning and the end of TITLE are indicated by "----". Other information follows the colon ":".

After the team members, there will be a line of twenty hyphens (--------------------). Please note that you may only provide the **LEADER** and **SUPPORTER**. The **MEMBER** can be None.

The next N lines consist of 6 characters each, ending with "------". These represent Runestones preparing to fall onto the stage. Your stage is under "------", and it consists of a 5x6 2D character array.

hw09+

Input

There are 6 types of runestones. In the input, uppercase letters represent Reinforced Runestones, and lowercase letters represent Normal Runestones:

- Water: W w
- Fire: F f
- Earth: E e
- Light: L l
- Dark: D d
- Health: H h

Data Spec

- $N \leq 10000$
- $len(\text{Monster name}) < 100$
- $0 \leq \text{Monster attack} < 10000$
- $0 \leq \text{Monster recovery} < 10000$
- $\text{Monster type} \in \{\text{"Water"}, \text{"Fire"}, \text{"Earth"}, \text{"Light"}, \text{"Dark"}\}$
- $\text{Monster LeaderSkill} \in \{\text{"DEFAULT"}, \text{"THREE\_ANY"}, \text{"TWO\_HEART\_\{TYPE\}"}\}$

hw09+

Output

Same as hw09

Hint

Same as the game, a team's dissolving strategy is effected by the **LEADER** and the **SUPPORTER**, you need to consider both of them.

For example, if the LEADER has THREE_ANY, and the SUPPORTER has TWO_HEART_FIRE, this means for heart and fire ruinestone can be dissolved by aligning 2 or more of them, the other types of ruinestone can be dissolved by aligning 3 or more of them

> **Tips**
> This is an extra homework, good luck.

Struct

- A set of variables

```
struct Monster {
  int id;
  char name[10];
  int health;
  int attack;
  int recovery;
  // ...
};
```

Struct

- A set of variables

```
struct Monster {
  int id;
  char name[10];
  int health;
  int attack;
  int recovery;
  // ...
};
```

```
struct Team {
  struct Monster monster[5];
  struct Monster helper;
};
```

Struct

Use `typedef`

```
typedef struct Monster Monster;
typedef struct Team Team;
```

Struct

Use `typedef`

```
typedef struct Monster Monster;
typedef struct Team Team;
```

Initialize like a normal variable

```
Monster monster_1;
monster_1.id = 10560;
strcpy(monster.name, "Gyre of Resonance - No
monster_1.health = 6094;
monster_1.attack = 2209;
monster_1.recovery = 237;
```

Struct

Use `typedef`

```c
typedef struct Monster Monster;
typedef struct Team Team;
```

Initialize like a normal variable

```c
Monster monster_1;
monster_1.id = 10560;
strcpy(monster.name, "Gyre of Resonance - No
monster_1.health = 6094;
monster_1.attack = 2209;
monster_1.recovery = 237;
```

Or using malloc

```c
Monster* monster_1 = malloc(sizeof(Monster))
monster_1 -> id = 10560;
strcpy(monster -> name, "Gyre of Resonance -
monster_1 -> health = 6094;
monster_1 -> attack = 2209;
monster_1 -> recovery = 237;
```

Struct

Construct a team

```
Team team;
team.helper = monster_1;

for(int i = 0; i < 5; i++) {
  team.monster[i] = monster_1
}
```

Struct

Construct a team

```
Team team;
team.helper = monster_1;

for(int i = 0; i < 5; i++) {
  team.monster[i] = monster_1
}
```

DFS and BFS

- Path Finding and Shortest Paths
- Connectivity Analysis
- and more senarios in future lessons

Graph Searching Algorithm Visualizer

Nerds: Huakun Shen, Yujie Miao

Board Size Setting

Width

Height

Apply

Simulator Speed

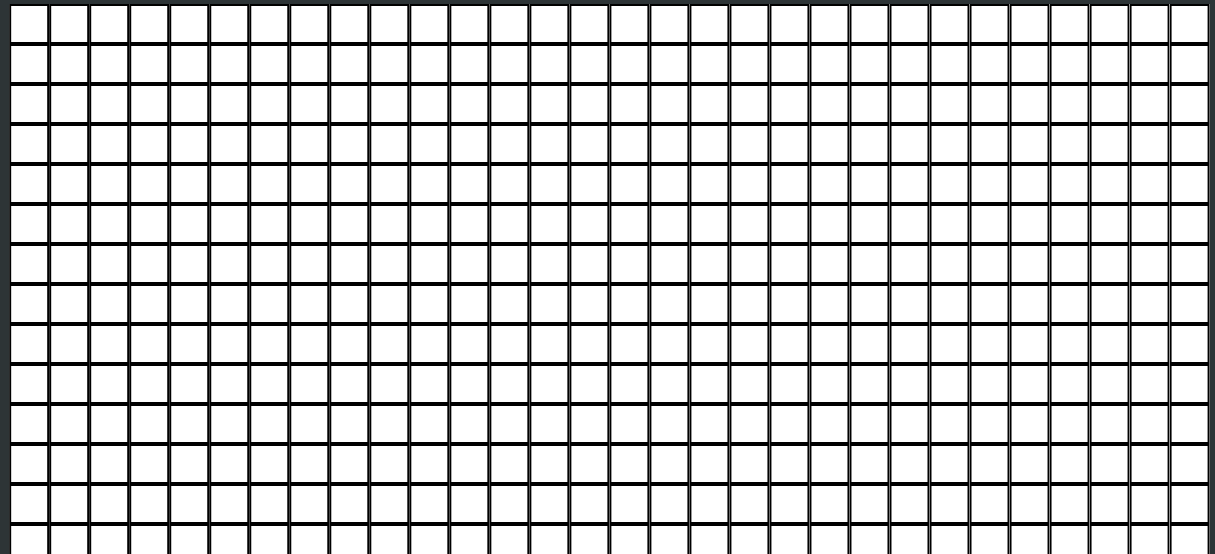
Frame Rate:
50

Log

Selecting Mode Wall Clear Source Target

Selecting Mode BFS DFS A*

Random Walls | Clear Board | Start Search



Depth-First Search (DFS)

- **Definition:** DFS explores as far as possible along each branch before backtracking
- **Steps:**
  1. Start from a source node
  2. Define your strategy. For example, left, up, right, down
  3. Rather peak the destination or step in, using recursive functions
  4. Mark visited nodes to avoid revisiting
  5. Returns when all strategy completes

Depth-First Search (DFS)

```
int dfs(int board[10][10], int i, int j) {
  if(dfs[i][j] == 0) {
    return 0;
  }
  int count = 0;
  if(i + 1 < 10) count += dfs(board, i + 1, j);
  if(i - 1 >= 0) count += dfs(board, i - 1, j);
  if(j + 1 < 10) count += dfs(board, i, j + 1);
  if(j - 1 >= 0) count += dfs(board, i, j - 1);

  return count;
}
```

Depth-First Search (DFS)

```
int dfs(int board[10][10], int i, int j) {
  if(dfs[i][j] == 0) {
    return 0;
  }
  int count = 0;
  if(i + 1 < 10) count += dfs(board, i + 1, j);
  if(i - 1 >= 0) count += dfs(board, i - 1, j);
  if(j + 1 < 10) count += dfs(board, i, j + 1);
  if(j - 1 >= 0) count += dfs(board, i, j - 1);

  return count;
}
```

- I prefer step in rather than peak.

Depth-First Search (DFS)

```
int dfs(int board[10][10], int i, int j) {
  if(dfs[i][j] == 0) {
    return 0;
  }
  int count = 0;
  if(i + 1 < 10) count += dfs(board, i + 1, j);
  if(i - 1 >= 0) count += dfs(board, i - 1, j);
  if(j + 1 < 10) count += dfs(board, i, j + 1);
  if(j - 1 >= 0) count += dfs(board, i, j - 1);

  return count;
}
```

- I prefer step in rather than peak.
- Remember check first to avoid out of boundary.

Depth-First Search (DFS)

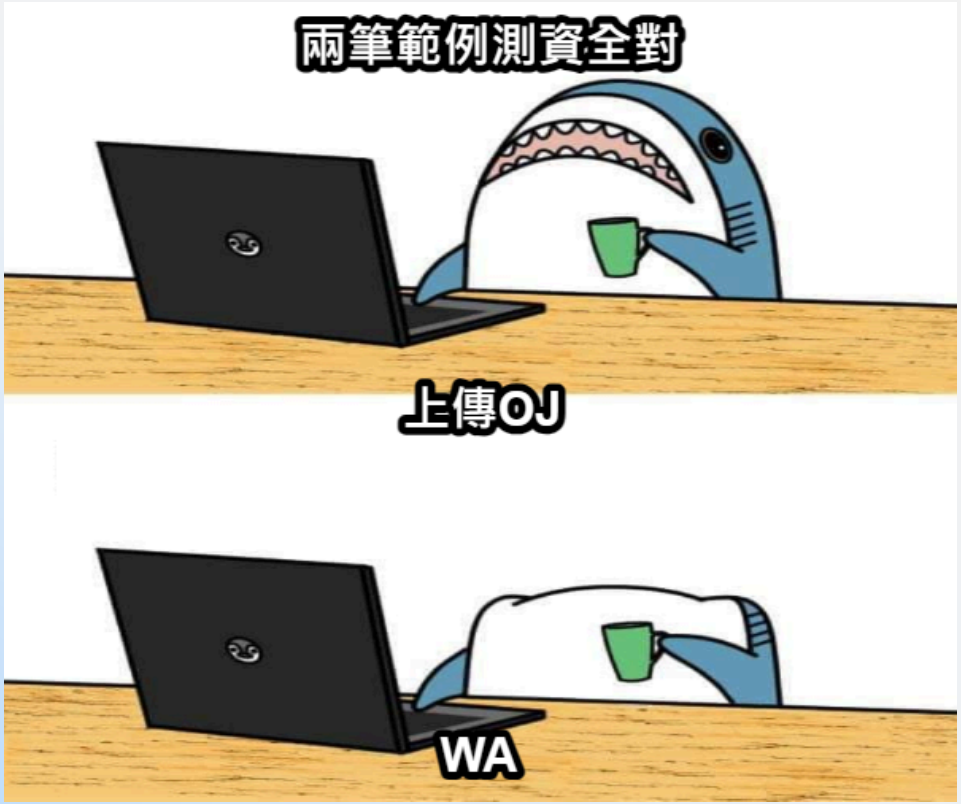
```
int dfs(int board[10][10], int i, int j) {
  if(dfs[i][j] == 0) {
    return 0;
  }
  int count = 0;
  if(i + 1 < 10) count += dfs(board, i + 1, j);
  if(i - 1 >= 0) count += dfs(board, i - 1, j);
  if(j + 1 < 10) count += dfs(board, i, j + 1);
  if(j - 1 >= 0) count += dfs(board, i, j - 1);

  return count;
}
```

- I prefer step in rather than peak.
- Remember check first to avoid out of boundary.
- Return value depends on what you need, which is important in recursive.

Breadth-First Search (BFS)

- **Definition:** BFS explores a graph level by level
- **Steps:**
  1. Start from a source node
  2. Explore all neighbors at the current depth **before** moving to the next depth
  3. Use a **queue** to keep track of nodes to visit
  4. Mark visited nodes to avoid revisiting
- We will learn this after we learn queue.

Debug strategies



Debug strategies

Debug strategies

- Instead of solely generating test cases on your own, consider trying alternative test cases that align with the problem’s specified range.

Debug strategies

- Instead of solely generating test cases on your own, consider trying alternative test cases that align with the problem's specified range.
- Be mindful that the array in the `main()` function has a limit, approximately **1 million integers**. If you require such a sizable array, consider either breaking it down or utilizing dynamic memory allocation with `malloc()`.

Debug strategies

- Instead of solely generating test cases on your own, consider trying alternative test cases that align with the problem's specified range.
- Be mindful that the array in the `main()` function has a limit, approximately **1 million integers**. If you require such a sizable array, consider either breaking it down or utilizing dynamic memory allocation with `malloc()`.
- Ensure that your code is easily readable, enabling others to understand what you have written.

Debug strategies

- Instead of solely generating test cases on your own, consider trying alternative test cases that align with the problem's specified range.
- Be mindful that the array in the `main()` function has a limit, approximately **1 million integers**. If you require such a sizable array, consider either breaking it down or utilizing dynamic memory allocation with `malloc()`.
- Ensure that your code is easily readable, enabling others to understand what you have written.
- Similarly, incorporating comments in your code aids others in comprehending your thought process and understanding the purpose behind each section.

Debug strategies

- Instead of solely generating test cases on your own, consider trying alternative test cases that align with the problem's specified range.
- Be mindful that the array in the `main()` function has a limit, approximately **1 million integers**. If you require such a sizable array, consider either breaking it down or utilizing dynamic memory allocation with `malloc()`.
- Ensure that your code is easily readable, enabling others to understand what you have written.
- Similarly, incorporating comments in your code aids others in comprehending your thought process and understanding the purpose behind each section.
- Google and ChatGPT can be valuable resources. Asking questions and understanding the responses is more beneficial than relying on mere copy-pasting.

Some Bad examples

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int a;
    int b[10001];
    while(cin >> a){
        int c = 0, d = 0, ans = 0;
        if(a == 0) break;
        for (int i = 0; i < a; i++)
        {
            cin >> b[i];
        }
        sort(b, b + a);
        // ...
    }
}
```

Some Bad examples

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int a;
    int b[10001];
    while(cin >> a){
        int c = 0, d = 0, ans = 0;
        if(a == 0) break;
        for (int i = 0; i < a; i++)
        {
            cin >> b[i];
        }
        sort(b, b + a);
        // ...
    }
}
```

Some Bad examples

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int a;
    int b[10001];
    while(cin >> a){
        int c = 0, d = 0, ans = 0;
        if(a == 0) break;
        for (int i = 0; i < a; i++)
        {
            cin >> b[i];
        }
        sort(b, b + a);
        // ...
    }
}
```

Some Bad examples

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int a;
    int b[10001];
    while(cin >> a){
        int c = 0, d = 0, ans = 0;
        if(a == 0) break;
        for (int i = 0; i < a; i++)
        {
            cin >> b[i];
        }
        sort(b, b + a);
        // ...
    }
}
```

Thanks for listening

Computer Programming 1

2023/12/9 Andy Hung

hw09 - Input

- Some of the input is unnecessary, use `fgets` to ignore it.
- Use `scanf()` wisely, like:

```
scanf("TYPE : %s\n", monster[i].type);
```

- Use `strcmp()` to stop reading queue
- Make sure you have right input before dealing with the problem, same for other problems.

hw09 - Hint

- Repeat until the board doesn't have dissolvable stones.
  1. Dissolve the board.
  2. Drop stones.
- Calculate stones.
- Sort.

hw09 – Output

I’ve gave you the answer.

hw09+ - Hint

> **Tips**
> Sorry for the sample, I've changed the sample output.

- Only **LEADER** and **SUPPORTER** will affect the strategy.
- Try to learn what OOP is, and take a look at your code again.
  - Maybe declaring a **function pointer** in your struct?
  - Maybe wrapping different functions to implent function overloading?