1. Thuật toán sừ dụng cho 2D pointer array trong 1 lần lặp đúng

Bước 1: Xác định xem cell được chọn nào trong 2 cell được chọn nằm ở phía sau cell còn lại (tức là có cột y lớn hơn), gọi là maxCell và minCell.

Bước 2: Sử dụng vòng lặp while để lặp hết *hàng* của maxCell từ vị trí (cột của maxCell + 1), trong mỗi vòng lặp gán “pokemon”, bắt đầu ở (cột của maxCell) bởi pokemon ở (cột của maxCell + 1). Vòng lặp dừng lại khi gặp cell có pokemon là ‘0’ hoặc là đến cuối hàng.

Bước 3: Sau khi kết thúc vòng lặp trên gán pokemon cho cell kết thúc (cell cùng hàng macCell và cell cuối cùng trong hàng có pokemon khác ‘0’) là ‘0’.

Bước 4: Làm tương tự với minCell.

Phân tích: Bằng cách chọn ra và bắt đầu ở cell nằm về bên phải cell còn lại, ta có thể bao quát được các trường hợp 2 cell được chọn nằm cùng hàng và có thể liên tiếp nhau. Bằng cách dùng vòng while ta có thể tiết kiệm thời gian xử lí thuật toán của thuật toán.

Độ phức tạp trong trường hợp tệ nhất trong “1 lần” chọn 2 cell đúng (trường hợp mới bắt đầu chơi và 2 cell được chọn nằm ở đầu 2 hàng khác nhau) : 1 phép so sánh để tìm ra cell max min, 2 vòng lặp while lặp đến cuối 2 hàng (của cell max và min) có tổng cộng 2\*(WIDTH - 1) lần lặp, và trong mỗi vòng lặp có 2 phép kiểm tra là i < WIDTH – 1 và cell cùng hàng, cột tiếp theo có pokemon != ‘0’, nên ta có tất cả 4 \* (WIDTH – 1) phép so sánh. Tổng cộng **4\*WIDTH – 3** phép so sánh tất cả trong trường hợp tệ nhất trong 1 lần.

(Độ phức tạp trong trường hợp tốt nhất trong 1 lần là chọn 2 cell nằm cuối: lúc này số phép so sánh là 1 + 2\*2 = 5.)

Độ phức tạp để hoàn thành cả màn chơi trong trường hợp tệ nhất (người chơi toàn chọn 2 ô đầu hàng ở 2 hàng khác nhau trong cả màn chơi): ví dụ Board có số cell là HEIGHT \* WIDTH, số lần chọn đúng cần là N = HEIGHT \* WIDTH / 2. Vậy có:

+ N phép so sánh tìm cell max, cell min.

+ Trong lần đầu tệ nhất số phép so sánh là 4 \* (WIDTH - 1), lần tiếp theo nếu giả sử vẫn chọn 2 cell đầu của vào 2 hàng đó thì số phép so sánh là 4 \* (WIDTH – 2), cứ vậy tới hết 2 hàng đó ta có 4 \* (WIDTH - 1)! phép so sánh.

+ Để tiện tính toán giả sử HEIGHT là chẵn thì số cặp hàng là HEIGHT / 2. 1 cặp hàng cần 4 \* (WIDTH – 1)! phép so sánh thì để so sánh cả bảng cần 2 \* HEIGHT \* (WIDTH - 1)! phép so sánh.

+ Tổng cộng: **N + 2\*HEIGHT\*(WIDTH-1)!** phép so sánh trong trường hợp tệ nhất cho 1 màn chơi.

Hàm toán tính thời gian được sử dụng là clock() trong ctime.

Run time thực tế trong 1 lần lặp tệ nhất là: 0,002s.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

Không tính run time cho cả màn chơi vì còn nhiều yếu tế không liên quan tới thuật toán đề cập như vẽ board lên console, tốc độ tay người chơi.

1. Thuật toán sử dụng linked list 1 chiều

Về các node trong link list thì mỗi node có 1 biến char chứa pokemon, 1 biến int chứa hàng x, 1 biến int chứa cột y, và 1 node\* pNext chỉ node tiếp theo. Nói rõ hơn về phép gán dữ liệu cho linked list này thì x = i / WIDTH, còn y = i % WIDTH khi ta lặp i = 0 đến i < HEIGHT \* WIDTH.

Các hàm phụ xài trong thuật toán:

- getNode nhận vào pHead và pos, trả về địa chỉ của node thứ pos

Về thuật toán trong 1 lần lặp đúng:

Bước 1: Xài hàm getNode để lấy địa chỉ 2 node được chọn.

Bước 2: Gán pokemon của 2 node trên là ‘1’.

Bước 3: Xài hàm getNode để lấy địa chỉ của 2 node đầu hàng của 2 node được chọn.

Bước 4: Duyệt 2 hàng có node được chọn bắt đầu từ đầu hàng và duyệt 2 lần, mỗi hàng 1 lần.

Bước 5: Khi gặp node có pokemon là ’1’. Gán pokemon của node đó là pokemon của node kế tiếp node đó trong trường hợp pokemon đó khác ‘0’ và node sau đó nằm cùng hàng.

Bước 6: Tiếp tục gán cho các node sau đó là pokemon của node kế sau đó cho đến cuối hàng.

Bước 7: Gán pokemon của node cuối hàng là ‘0’.

Phân tích:

+ Vì linkedlist 1 chiều nên ta không có cách nào để lặp ngược lại từ cuối hàng đến vị trí node được chọn để gán.

+ Nên ta cần lặp 2 lần cho 2 hàng có node được chọn (1 lần 1 hàng và lưu ý là 2 hàng này thực chất có thể cùng là 1 hàng) và lặp từ đầu hàng đến cuối hàng.

+ Bằng cách đặt trước pokemon của node được chọn là ‘1’, ta có thể phân biệt với ‘0’ để tiết kiệm số lần phải lặp, đồng thời trong trường hợp 2 node được chọn cùng hàng thì vẫn sẽ có thể tìm ra node chứa pokemon là ‘1’ vì ta lần lượt lặp từ đầu hàng đến cuối hàng.

+ Không sử dụng xóa node vì rất khó canh vừa với mảng 2 chiều, vì sau khi thao tác trên linklisted ta cần chép lại vào 2D pointer array để hiển thị trên console cũng như là sử dụng lại các chứa năng di chuyển con trỏ trên mảng 2 chiều.

Độ phúc tạp trong trường hợp tệ nhất trong 1 lần chọn đúng:

* Vì trong bất kỳ lần chọn nào ta cũng cần duyệt 2 lần từ đầu hàng đến cuối hàng để tìm node có pokemon là ‘1’ nên trường hợp tệ nhất là chọn 2 node ở cuối 2 hàng khác nhau.
* Lúc này số lần duyệt mỗi hàng là WIDTH – 1. Hai hàng là 2 \* (WIDTH – 1).
* Mỗi lần duyệt cần 4 phép so sánh, 3 phép là ((node->pNext != NULL && node->pNext->x == node->x) || node->y == WIDTH – 1). Lưu ý node->y == WIDTH – 1 để xác định khi đã duyệt đến cuối hàng. 1 phép còn lại để kiểm tra xem pokemon của node có bằng ‘1’.
* Vậy số phép so sánh trong trường hợp tệ nhất là **8 \* (WIDTH – 1)**

Độ phức tạp trong trường hợp tốt nhất (chọn 2 node đầu hàng) là: **4 \* (WIDTH – 1)** vì ta có thể bỏ 2 phép so sánh là node->y == WIDTH – 1 và phép kiểm tra == ‘1’ so với trường hợp trên vì vẫn cần phải lặp đến cuối hàng để gán tuần tự.

Độ phức tạp cho cả màn chơi trong trường hợp tệ nhất (chọn toàn 2 node cuối cùng).

* Lần chọn đầu cần 8 \* (WIDTH – 1) phép so sánh, lần kế tiếp giả sử vẫn chọn 2 node cuối cùng trong 2 hàng đó cần 8 \* (WIDTH – 2) + 2 phép so sánh, lần kế nữa là 8 \* (WIDTH – 3) + 4, cứ vậy đến hết 2 hàng đó ta cần tất cả 8 \* (WIDTH – 1)! + (WIDTH-1)\*(WIDTH-2) phép so sánh.

(Lưu ý lý do ta + 2\*i ở đây là vì sau khi tìm ra node có pokemon là ‘1’ thì thuật toán sẽ tiếp tục lặp đến cuối hàng để gán tuần tự)

* Vẫn giả sử có HEIGHT / 2 cặp hàng thì ta cần tất cả  **(HEIGHT / 2) \* (8 \* (WIDTH – 1)! + (WIDTH-1)\*(WIDTH-2))** phép so sánh trong trường hợp tệ nhất cho 1 game với thuật toán trên.

Run time thực tế cho 1 lần chọn 2 node cuối cùng mỗi hàng: ~ 0s

Text

Description automatically generated

A screenshot of a computer

Description automatically generated