بسم الله الرحمن الرحیم

ما در این فایل همه چیز را به ۴ بخش تقسیم میکنیم(هر بخش برای یک کد هست) و داخل آن به پاسخ سوالات و توضیحات راجب نتایج بدست امده و... می پردازیم.

بخش اول(کد ۱)

سوال ۱: در چند درصد مواقع توانستید پاسخ کامل را بیابید؟

پاسخ: همانطور که با اجاری کد اول متوجه میشویم، در حدود ۲۰ درصد مواقع(چون ما آرایه اولیه را بصورت رندوم تولید میکنیم، هر بار ممکن است این اعدد متفاوت از دفعه قبل باشد، اما جواب ها در حدود همین مقدار هستند).

سوال ۲: بطور متوسط چند مرحله برای یافتن پاسخ کافی بوده است؟

پاسخ: یک شمارنده در تابع تپه نوردی میگذاریم (بیرون از حلقه بی نهایت) و انرا صفر میکنیم، با هر بار اجرای این حلقه، این شمارنده یک واحد زیاد میشود، بعد از اینکه کار تابع تمام شد و حاصل را برگرداند، ما چک میکنیم که آیا در پاسخ داده شده،(که شامل یک حل و یک شمارنده است) در حلی که به ما داده است، آیا این حل، حلی ایمن هست یا خیر(حل ایمن حلی است که هیچ دو وزیری هم دیگر را تهدید نکند) اگر حلی ایمن بود حاصل شمارنده را با یک متغیر به اسم "جمع(sum)" جمع میزنیم و انرا درون متغیر "جمع" میریزیم، در اخر برای اینکه تعداد متوسط مراحل را برای حل پیدا کنیم، این عدد را تقسیم بر تعداد موفقیت ها(که یک شمارنده جدا هست که تعداد کل حل های ایمن را میشمارد) میکنیم، که در این کد

حاصل عدد ۴ شده است(چون ما آرایه اولیه را بصورت رندوم تولید میکنیم، هر بار ممکن است این اعدد متفاوت از دفعه قبل باشد، اما جواب ها در حدود همین مقدار هستند)

نکته: کد را بصورت کامل هم در ویدئو و هم بصورت کاملا منظم در خود فایل کامنت گذاری کرده ام، اما با اینحال یک مختصری از کد میگویم، در این کد(که الگوریتم تپه نوردی را پیاده سازی کرده ایم) ما سعی بر این داریم که وضعیت فعلی خود را بهبود ببخشیم و هر بار که ماکسیمم مقدار ارزش همسایه ها را بدست می اوریم و با ارزش نود فعلی مقایسه میکنیم اگر به جایی رسیدیم که دیگر نود فعلی بهترین یا مساوی بهترین همسایه بود(یعنی همه همسایه ها وضعیت و ارزش بد تری نسبت به نود فعلی دارند یا مساوی با ان هستند) انرا بازمیگردانیم.

همچنین در این کد واضح است که اگر به یک سطح هموار(flat) رسیدیم دیگر ادامه نمیدهیم و همین نود فعلی را برمیگردانیم.

اگر هم ارزش نود فعلی اکیدا کمتر بهترین نود همسایه بود، نود همسایه را درون نود فعلی میریزیم

بخش ۲(کد دوم):

سوال ۱: در چند درصد مواقع توانستید پاسخ کامل را بیابید؟

پاسخ: همانطور که با اجاری کد دوم متوجه میشویم، در حدود ۴۵ درصد مواقع(چون ما آرایه اولیه را بصورت رندوم تولید میکنیم، هر بار ممکن است این اعدد متفاوت از دفعه قبل باشد، اما جواب ها در حدود همین مقدار هستند)

سوال ۲: بطور متوسط در چند مرحله پاسخ رایافتید؟

پاسخ: یک شمارنده درون تابع تپه نوردی میگذاریم (بالای حلقه بی نهایت) و انرا صفر میکنیم، با هر بار اجرای این حلقه، این شمارنده یک واحد زیاد میشود، بعد از اینکه کار تابع تمام شد و حاصل را برگرداند، ما چک میکنیم که آیا در پاسخ داده شده،(که شامل یک حل و یک شمارنده است) در حلی که به ما داده است، آیا این حل، حلی ایمن هست یا خیر(حل ایمن حلی است که هیچ دو وزیری هم دیگر را تهدید نکند) اگر حلی ایمن بود حاصل شمارنده را با یک متغیر به اسم "جمع(sum)" جمع میزنیم و انرا درون متغیر "جمع" میریزیم، در اخر برای اینکه تعداد متوسط مراحل را برای حل پیدا کنیم، این عدد را تقسیم بر تعداد موفقیت ها(که یک شمارنده جدا هست که تعداد کل حل های ایمن را میشمارد) میکنیم، که در این کد حاصل عدد **۷** شده است(چون ما آرایه اولیه را بصورت رندوم تولید میکنیم، هر بار ممکن است این اعدد متفاوت از دفعه قبل باشد، اما جواب ها در حدود همین مقدار هستند)

همانطور که در بخش قبل(اول) گفته ام، کد ها را تماما کامنت گذاری کرده ام و ویدئویی برای توضیح انها نیز تهیه کرده ام، اما به توضیح مختصری راجب این کد میپردازیم

در این کد که الگوریتم تپه نوردی را پیاده سازی کرده ایم ، خیلی شبیه به کد سوال ۱ است که در بالاتر انرا نیز توضیح داده ایم، اما تفاوت هایی با کد ۱ دارد، در کد ۱ اگر ارزش همه همسایه ها کمتر یا مساوی با ارزش نود فعلی بود، ما آن خانه(نود فعلی) را به عنوان پاسخ برمیگرداندیم، اما در اینجا اگر ارزش نود فعلی اکیدا بزرگتر از بهترین نود همسایه بود جواب را برمیگردانیم(نود فعلی) ولی اگر مساوی بود تا ۴۰ بار برای مثال اجازه حرکت به انرا

میدهیم(یعنی اگر به یک سطح هموار رسیدیم، تا مقداری اجازه حرکت داریم) و برای حالت دیگر هم مشابه کد ۱ هست.

بخش سوم(کد سوم):

سوال ۱:

پاسخ: (۹۵ درصد) توضیحات اضافه دقیقا مشابه با دو مورد قبل است.

سوال ۲:

پاسخ: (۵۰۱ مرحله) توضیحات اضافه دقیقا مشابه با دو مورد قبل است.

نکته مهم: نکته ای که میتوان از این بخش متوجه شد این است که من چون تابعی را که زمان را تبدیل به دما میکند، جوری نوشته ام که به تعداد زیادی اجرا میشود، این دقت بالا و مرحله زیاد بدست امده، اگر برای مثال ضرب در ۱۰۰ داخل این تابع را به ضرب در ۵ تبدیل کنم، پاسخ برای سوال اول و دوم به ترتیب به (حدودا) ۲۰ درصد و ۲۶ مرحله کاهش پیدا میکند.

توضیحات مختصر راجب کد: در این کد با استفاده از الگوریتم سرد شدن شبیه سازی پیاده سازی شده است، ما ابتدا در یک دمای بالا(داغ) هستیم و با گذشت زمان میدانیم این دما کاهش میابد، حال در یک حلقه بی نهایت زمان را به تابع میدهیم و منتظر میمانیم تا وقتی که

دما صفر بشود تا نود فعلی(کارنت) را برگردانیم، و هر بار ارزش نودی که تولید شده (با استفاده از نود فعلی) است را از ارزش نود فعلی کم میکنیم اگر این مقداری مثبت بود نود تولید شده را درون نود فعلی میریزم، واگر این شرط برقرار نبود با یک احتمالی اینکار را انجام میدهیم.

بخش چهارم(کد چهارم):

در این بخش به توضیح کلی درباره خود کدها و سپس بطور مفصل ۲ تابع که کمتر کامنت گذاری کرده ایم میپردازیم.

در این کد که کمی پیچیده تر از بقیه کدهاست و سعی کردم بیشتر کامنت گذاری بکنم(جز برای دوتابع) دیدن کامنت ها بسیار مفید تر هست، اما اگر بخواهیم خلاصه ای از این الگوریتم بگویم به این نحو هست که ابتدا ما راس اولیه را به تابع AND_OR_SEARCH پاس میدهیم که ان نیز تابع OR_SEARCH را صدا میزند و ان نیز تابع AND_SEARCH را صدا میزند و خود همین تابع، تابع OR_SEARCH را صدا میزند(یک حالت یکی در میان طور دارد، منظور این هست که اگر به گراف نگاه کنیم، در یک سطح OR ها و در سطح بعدی AND ها هستند و تا رسیدن به برگ ها همینطور ادامه دارد). و چیزی که این تابع(AND_OR_SEARCH) برمیگرداند ممکن است یک جواب(بهتر است بگوییم: ممکن است مجموعه جواب برگرداند) یا یک شکست برمیگرداند.

نحوه عملکرد تابع ACTIONS:

این تابع ۱ ورودی میگیرد که همان وضعیت هست(شامل وضعیت خانه ۱، وضعیت خانه ۲، محل قرار گیری جارو برقی) اگر خانه چپ کثیف باشد یا تمیز باشد وجارو هم در خانه چپ باشد، میتوان عمل مکش را به لیست اضافه کرد(۲ حالت)، اگر هم خانه راست کثیف باشد و یا تمیز باشد و جارو هم در خانه راست باشد میتوان باز هم عمل مکش را به لیست اضافه کرد(۲ حالت).

اگر در خانه چپ باشیم میتوانیم به خانه راست برویم(اضافه کردن حرکت راست) و برعکس(۲ حالت)

درآخر لیست به دست امده را برمیگردانیم.

نحوه عملکرد تابع RESULTS:

این تابع ۲ ورودی میگیرد، یکی وضعیت و دیگری عمل(مکش و..)، به این نحو که **فرض کنید ما در خانه سمت چپ هستیم**، ۳ حالت پیش می آید، **اول اینکه عمل مکش انجام شود:**

در خانه چپ زباله باشد میتوان انرا مکش کرد و به وضعیت خانه راست دست نزد، میتوان هم اگر در خانه راست هم زباله بود هر دو را مکش کرد.

اگر هم زباله ای در خانه سمت چپ نباشد(طبیعتا باید در خانه راست زباله باشد وگرنه این خانه هدف هست!!) میتوانیم در همین وضعیت باقی ماند، یا اینکار میتواند زباله ای را از بیرون وارد این خانه بکند.

عمل راست رفتن اتفاق بیافتد:

به وضعیت خانه ها دست نمیزنیم و فقط به سمت راست میرویم و موقعیت را به "راست" تغیر میدهیم

عمل چپ رفتن اتفاق بیافتد:

این عمل امکان پذیر نیست و در همین خانه باید بمانیم(برای اینکه همه حالات را نظر بگیریم نوشته شده است).

و بطور مشابه برای وقتی که در خانه سمت راست باشیم

سپس بعد از اینکار ها را انجام دادیم، در هر حالت، خروجی بدست امده از ان را به لیستی اضافه میکنیم و در اخر ان لیست را برمیگردانیم.

علی طاهری

۴۰۰۱۲۳۲۳