مستندات پروژه اول درس طراحی الگوریتم

اعضای تیم: عرفان عباسی، گیتا قاسمی

ترم: پاییز 1403

مقدمه:

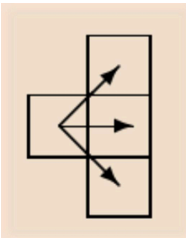
هدف از این پروژه، بیشینه کردن مقدار خوشحالی کریم باستانی است. این بستنی‌فروش با عبور از خانه‌های ماتریسی n*m مقدار خوشحالی‌اش افزایش پیدا می‌کند. پروژه سه فاز دارد و هر فاز با استفاده از دو الگوریتم Greedy و Dynamic Programming حل می‌شود.

ساختار پروژه:

در روت اصلی پروژه یک فولدر به نام **src** وجود دارد که منطق الگوریتم و پروژه در آن قرار دارد. در داخل این فولدر، سه دایرکتوری به نام‌های **phase1**، **phase2** و **phase3** وجود دارد. در هر کدام از این دایرکتوری‌ها یک فایل **main** قرار دارد که الگوریتم مربوط به هر فاز در آن نوشته شده است. هر فاز به صورت یک کلاس پیاده‌سازی شده و تمامی فازها از یک اینترفیس مشترک ارث‌بری کرده‌اند. این اینترفیس در فولدر **interfaces** قرار دارد و متدهایی را تعریف می‌کند که هر فاز باید آن‌ها را پیاده‌سازی کند.

با اجرای فایل **main** هر فاز، می‌توان ورودی را از طریق کنسول به آن داد و خروجی را مشاهده کرد. همچنین می‌توان همه فازها را به طور همزمان اجرا کرد. برای این کار، تست کیس‌ها برای هر فاز در فایل **sample_input** نوشته می‌شوند. سپس با اجرای فایل **main** در روت اصلی پروژه، تمام فازها تست کیس‌ها را از فایل **sample_input** می‌خوانند و خروجی‌های مربوطه را در فایل‌های **sample_output** هر فاز می‌نویسند.

توضیح درباره هر فاز پروژه (فاز ۱، ۲، ۳):



کریم در ابتدای کار می‌تواند بستنی‌هایش را در هر کدام از شهرهای ستون اول (از سمت چپ) بارگیری کند. در ادامه کریم می‌تواند با حرکت از شهری که در آن قرار دارد به هر کدام از سه شهر مجاور آن در ستون سمت راست حرکت کند. (مطابق شکل)

مسیر حرکت او تا سمت راست‌ترین ستون ادامه خواهد داشت و در آن ستون متوقف خواهد شد. کار ما در هر کدام از فازها پیدا کردن مسیری است که کریم را خوشحال‌تر از هر مسیر دیگر بکند و همچنین چالش‌های دیگری که کریم در این راه دارد را نیز حل و فصل کند.

فاز اول:

در این فاز چالشی در مسیر کریم وجود ندارد. صرفا باید مسیری که مجموع جمعیت‌های شهرهای عبوری آن بیشینه است را پیدا کنید.

ورودی:

- یک ماتریس با ابعاد m×n که مقدار شادی (ارزش) هر خانه را نشان می‌دهد.
- کاربران ابعاد ماتریس و مقادیر را به صورت دستی وارد می‌کنند.

در خط اول دو عدد n و m است که ابعاد شهر می‌باشند. سپس در n خط بعدی هر کدام m عدد نامنفی ورودی داده میشود که با یک فاصله از هم جدا شده اند.

خروجی:

- مقدار حداکثر شادی قابل دستیابی.
- مسیر طی‌شده در ماتریس برای رسیدن به این مقدار.

در خط اول خوشحالی کریم در مسیر بیشینه و درخط بعدی m عدد که نشان دهنده مسیری بهینه است.

توضیح دو روش:

1. **روش حریصانه:**

الگوریتم حریصانه از **ستون اول ماتریس** شروع می‌کند و خانه‌ای که بیشترین مقدار (شادی) را دارد، انتخاب می‌کند. سپس به ترتیب از هر ستون به ستون بعدی حرکت

می‌کند و به دنبال با ارزش‌ترین خانه در همسایگی خانه فعلی می‌گردد. همسایگان شامل سه خانه‌ی ممکن در ستون بعدی هستند:

- خانه‌ی بالایی (مورب به سمت بالا)
- خانه‌ی روبه‌رو (مستقیم در همان ردیف)
- خانه‌ی پایینی (مورب به سمت پایین)

این فرآیند تا رسیدن به آخرین ستون ادامه پیدا می‌کند و در نهایت **مقدار کل شادی و مسیر طی‌شده** را به عنوان خروجی بازمی‌گرداند.

پیچیدگی زمانی:

برای هر ستون(m)

برای هر سطر(n)

بیشترین مقدار از بین سه همسایگی آن را انتخاب می‌کند $O(1)$.

بنابراین پیچیدگی زمانی کل $O(m \times n)$ می‌شود.

2. روش برنامه‌ریزی پویا:

ایجاد جدول DP:
جدولی به ابعاد ماتریس ورودی ایجاد می‌کنیم که در هر خانه، بیشترین مقدار شادی قابل دستیابی تا آن خانه ذخیره می‌شود.

مقداردهی اولیه:
مقادیر ستون اول جدول DP را برابر با مقادیر ستون اول ماتریس ورودی قرار می‌دهیم.

محاسبه مقادیر هر خانه:
برای هر خانه از ستون دوم به بعد، بیشترین مقدار ممکن را از همسایگان ستون قبلی محاسبه کرده و به مقدار خانه‌ی فعلی در ماتریس ورودی اضافه می‌کنیم:

- همسایه‌ی بالا سمت چپ (در صورت وجود)
- همسایه‌ی مستقیم سمت چپ
- همسایه‌ی پایین سمت چپ (در صورت وجود)

پیدا کردن مقدار بیشینه:
پس از پر کردن جدول DP، بیشترین مقدار در ستون آخر جدول، مقدار بیشینه شادی است.

استخراج مسیر:
از خانه‌ای که مقدار بیشینه در ستون آخر قرار دارد، به عقب برمی‌گردیم و مسیری را که اختلاف آن با مقادیر خانه‌های همسایه برابر با مقدار اولیه ماتریس ورودی است، پیدا می‌کنیم. این مسیر، مسیر بهینه است.

پیچیدگی زمانی:

مشابه روش حریصانه، پیچیدگی زمانی $O(m \times n)$ است، اما از نظر حافظه به جدول DP نیاز دارد که این موضوع در ماتریس‌هایی با ابعاد بالاتر ممکن است از نظر مصرف حافظه چالش‌زا باشد.

بررسی بهینگی:

1. **روش حریصانه:**
   روش حریصانه همیشه بهترین مسیر ممکن را پیدا نمی‌کند.
   مثالی از شکست این روش زمانی است که خانه‌ای با مقدار شادی کمتر در ابتدا انتخاب شود، اما در ادامه مسیر به شادی بیشتری منجر شود.

2. **برنامه‌ریزی پویا:**
   برنامه‌ریزی پویا همیشه مسیر بهینه را پیدا می‌کند، چون تمام گزینه‌های ممکن را بررسی کرده و بهترین مسیر را در نظر می‌گیرد.

تحلیل و بررسی یک مثال:

Example:

Input:

```
[
   [1, 2, 3, 4, 5, 6, 7, 8],
   [2, 3, 4, 5, 6, 7, 8, 77],
   [3, 4, 5, 6, 7, 8, 9, 10],
   [4, 5, 6, 7, 8, 9, 10, 11],
   [5, 6, 7, 8, 9, 10, 11, 12],
   [6, 7, 8, 9, 10, 11, 12, 13],
]
```

Greedy Approach:

Max Happiness: 76

Path: [6, 6, 6, 6, 6, 6, 6, 6]

Time Complexity: O(m * n), m=6, n=8, 0.038 milliseconds

Memory Usage: 0.086 KB

DP Approach:

Max Happiness: 134

Path: [6, 6, 6, 6, 5, 4, 3, 2]

Time Complexity: O(m * n), m=6, n=8, 0.048 milliseconds

Memory Usage: 0.250 KB

مراحل اجرا (حریصانه):

1. **شروع از ستون اول:**
   - مقادیر ستون اول: `[1, 2, 3, 4, 5, 6]`
   - انتخاب خانه با بیشترین مقدار: `6` (ردیف 6)

2. **بررسی همسایگان ستون دوم:**
   - همسایگان خانه (6, 1) در ستون دوم:
     - بالا راست : 6
     - روبه‌رو: 7
   - انتخاب مقدار بیشینه: 7 (ردیف 6)

3. **بررسی همسایگان ستون سوم:**
   - همسایگان خانه (6, 2) در ستون سوم:
     - بالا راست: 7
     - روبه‌رو: 8
   - انتخاب مقدار بیشینه: 8 (ردیف 6)

4. **ادامه الگوریتم برای ستون‌های بعدی:**
   - ستون چهارم: انتخاب مقدار 9 (ردیف 6)
   - ستون پنجم: انتخاب مقدار 10 (ردیف 6)
   - ستون ششم: انتخاب مقدار 11 (ردیف 6)
   - ستون هفتم: انتخاب مقدار 12 (ردیف 6)
   - ستون هشتم: انتخاب مقدار 13 (ردیف 6)

نتیجه حریصانه:

- **مجموع شادی:** 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 = 76
- **مسیر:** [6, 6, 6, 6, 6, 6, 6, 6]

روش برنامه‌ریزی پویا (Dynamic Programming):

مراحل اجرا:

1. ساخت جدول DP
2. مقداردهی اولیه (ستون اول)

مقادیر ستون اول به عنوان مقدار اولیه dp استفاده می‌شوند:

```
dp = [
  [1, 0, 0, 0, 0, 0, 0, 0],
  [2, 0, 0, 0, 0, 0, 0, 0],
  [3, 0, 0, 0, 0, 0, 0, 0],
  [4, 0, 0, 0, 0, 0, 0, 0],
  [5, 0, 0, 0, 0, 0, 0, 0],
  [6, 0, 0, 0, 0, 0, 0, 0],
]
```

3. محاسبه مقادیر ستون به ستون:

- برای هر خانه `(i, j)`، جمع مقدار بیشینه از همسایگان ستون قبلی با ارزش خانه‌ی فعلی محاسبه می‌شود:

محاسبات برای هر ستون:

اعدادی که با رنگ سبز مشخص شده‌اند ارزش خانه‌ی فعلی هستند.

اعدادی که با قرمز مشخص شده‌اند بیشترین ارزش همسایگان ستون قبلی هستند.

ستون دوم:

```
dp = [
```

[1, 2 + 2 = 4, 0, 0, 0, 0, 0],

[2, 3 + 3 = 6, 0, 0, 0, 0, 0],

[3, 4 + 4 = 8, 0, 0, 0, 0, 0],

[4, 5 + 5 = 10, 0, 0, 0, 0, 0],

[5, 6 + 6 = 12, 0, 0, 0, 0, 0],

[6, 6 + 7 = 13, 0, 0, 0, 0, 0],

]

ستون سوم:

dp = [

[1, 4, 6 + 3 = 9, 0, 0, 0, 0, 0],

[2, 6, 8 + 4 = 12, 0, 0, 0, 0, 0],

[3, 8, 10 + 5 = 15, 0, 0, 0, 0, 0],

[4, 10, 12 + 6 = 18, 0, 0, 0, 0, 0],

[5, 12, 13 + 7 = 20, 0, 0, 0, 0, 0],

[6, 13, 13 + 8 = 21, 0, 0, 0, 0, 0],

]

محاسبات برای بقیه ستون‌ها به همین ترتیب ادامه دارد...

4. **استخراج نتیجه نهایی:**

مقدار بیشینه در آخرین ستون:

dp = [

[1, 4, 9, 16, 25, 36, 48, 61]

[2, 6, 12, 20, 30, 41, 53, **134**]

[3, 8, 15, 24, 34, 45, 57, 70]

[4, 10, 18, 27, 37, 48, 60, 73]

[5, 12, 20, 29, 39, 50, 62, 75]

[6, 13, 21, 30, 40, 51, 63, 76]

]

مقدار بیشینه: 134

حال برای پیدا کردن مسیر، پس از یافتن مقدار بیشینه در آخرین ستون، مراحل زیر را طی می‌کنیم:

1. **پیدا کردن خانه‌ی بیشینه در آخرین ستون از جدول DP:**
   در اینجا، بیشترین مقدار 134 است که در ستون آخر و ردیف دوم قرار دارد. شماره ردیف این خانه را در یک آرایه ذخیره می‌کنیم.
2. **بررسی همسایگان ستون قبلی:**
   اختلاف مقدار خانه‌ی فعلی با تک‌تک همسایگان ستون قبلی را محاسبه می‌کنیم. هر همسایه‌ای که اختلافش با خانه‌ی فعلی برابر با مقدار خانه در ماتریس ورودی باشد (ماتریسی که به الگوریتم داده شده است)، نشان می‌دهد که خانه‌ی قبلی همان خانه‌ای است که ما از آن به خانه‌ی فعلی رسیده‌ایم.
3. **تکرار مراحل 2 و 3 تا ستون اول:**
   این مراحل را ادامه می‌دهیم تا به ستون اول برسیم. در هر مرحله، شماره ردیف خانه‌ی انتخاب‌شده را در آرایه‌ای ذخیره می‌کنیم.
4. **جواب نهایی:**
   در پایان، آرایه‌ای که به‌دست آمده است، مسیر بهینه‌ای را نشان می‌دهد که برای رسیدن به بیشترین مقدار طی کرده‌ایم.

مراحل پیدا کردن مسیر (Path) به طور گام به گام:

ماتریس ورودی (شادی‌های هر خانه):

[

```
    [1, 2, 3, 4, 5, 6, 7, 8],
    [2, 3, 4, 5, 6, 7, 8, 77],
    [3, 4, 5, 6, 7, 8, 9, 10],
    [4, 5, 6, 7, 8, 9, 10, 11],
    [5, 6, 7, 8, 9, 10, 11, 12],
    [6, 7, 8, 9, 10, 11, 12, 13]
]
```

ماتریس DP (ماتریس بیشترین شادی تا هر نقطه):

```
[
    [1, 4, 9, 16, 25, 36, 48, 61],
    [2, 6, 12, 20, 30, 41, 53, 134],
    [3, 8, 15, 24, 34, 45, 57, 70],
    [4, 10, 18, 27, 37, 48, 60, 73],
    [5, 12, 20, 29, 39, 50, 62, 75],
    [6, 13, 21, 30, 40, 51, 63, 76]
]
```

مرحله 1: یافتن مقدار بیشینه در آخرین ستون

در ستون آخر (ستون 7)، بیشترین مقدار متعلق به خانه‌ای است که مقدار آن 134 است و در ردیف 1 قرار دارد (یعنی `(2, 7)` که در واقع ردیف دوم و ستون هفتم است). بنابراین، نقطه شروع ما در این خانه قرار دارد.

مرحله 2: شروع از خانه 134

ما از خانه‌ای که مقدار 134 در آن قرار دارد شروع می‌کنیم. این خانه در ردیف 1 و ستون 7 است.

مرحله 3: یافتن همسایگان در ستون قبلی

برای خانه‌ای که در ستون 7 و ردیف 1 قرار دارد، باید به دنبال همسایگان آن در ستون 6 بگردیم و اختلاف آن‌ها با خانه فعلی را بررسی کنیم.

همسایگان در ستون 6:

- خانه در ردیف 0: مقدار 48
- خانه در ردیف 1: مقدار 53
- خانه در ردیف 2: مقدار 57

حال اختلاف این مقادیر را از 134 محاسبه می‌کنیم:

- 134 - 48 = 86
- 134 - 53 = 81
- 134 - 57 = 77

در ماتریس ورودی، خانه‌ای که دارای مقدار 77 است همان خانه‌ای است که از آن به خانه 134 رسیدیم. این خانه در ردیف 1 و ستون 6 قرار دارد (مقدار 57 در آن خانه است).

مرحله 4: ادامه پیدا کردن مسیر با خانه 57

حال که در خانه 57 در ستون 6 هستیم، باید همسایگان این خانه را در ستون 5 بررسی کنیم.

همسایگان در ستون 5:

- خانه در ردیف 0: مقدار 41
- خانه در ردیف 1: مقدار 45
- خانه در ردیف 2: مقدار 48

- حال اختلاف این مقادیر از 57 را محاسبه می‌کنیم:

- 57 - 41 = 16
- 57 - 45 = 12
- 57 - 48 = 9

در ماتریس ورودی، خانه‌ای که دارای مقدار 9 است همان خانه‌ای است که از آن به خانه 57 رسیدیم. این خانه در ردیف 2 و ستون 5 قرار دارد (مقدار 48 در آن خانه است).

این مرحله را تا رسیدن به ستون اول ادامه میدهیم.
ایندکس‌هایی که در آرایه ذخیره کرده بودیم را به عنوان جواب نمایش میدهیم.

[6, 6, 6, 6, 5, 4, 3, 2]

فاز دوم:

در این فاز هوا بسیار گرم است. بستنی‌های کریم در میانه مسیر در خطر آب شدن قرار دارند. بعضی از ستون‌ها در کشور وجود دارند که در خانه‌های آن حروف X and O قرار دارد. کریم باید با عبور از یکی از خانه‌های Oدار آن ستون، که در واقع دارای فریزرهایی هستند که بستنی های کریم را دوباره سرد می‌کنند، از آب شدن بستنی‌های خود جلوگیری کند.

ورودی:

ورودی این فاز نیز مثل فاز قبلی‌ست با این تفاوت که در تعدادی از ستون‌ها، خانه‌ها به جای اعداد، حروف X و یا O دارند.
ورودی به شکل ماتریسی با ابعاد m * n شامل موارد زیر می‌باشد:
مقادیر عددی برای ارزش هر خانه.
نماد X برای سلول‌های مسدود شده.
نماد O برای نقاط یخچال.
کاربران باید ماتریس را ردیف به ردیف وارد کنند.

خروجی:

- حداکثر مقدار شادی قابل دستیابی.
- مسیر طی‌شده با اجتناب از سلول‌های مسدود شده و در نظر گرفتن نقاط یخچال.

تفاوت‌های کلیدی این فاز:

سلول‌های خاص:

سلول‌های مسدود شده (Blocked Cells): در مسیر قابل انتخاب نیستند.

نقاط یخچال (Refrigeration Points): ارزش خاصی دارند و در مسیر محاسبه می‌شوند.

عملکرد و محدودیت‌ها:

الگوریتم باید از انتخاب سلول‌های مسدود شده اجتناب کند.

در روش حریصانه، از یک مکانیزم "بازگشت" (*rollback*) برای بازگشت از انتخاب‌های نامناسب استفاده می‌شود.

توضیح دو روش:

1. **روش حریصانه:**

انتخاب سلول با بیشترین مقدار از بین همسایگان. اضافه کردن آن به یک پشته(stack)

اگر انتخاب مناسبی وجود نداشت، به عقب بازمی‌گردد (rollback).

عدد انتخاب شده را از پشته حذف میکند.

پیچیدگی زمانی:

در بدترین حالت، بازگشت‌ها و بررسی همسایگان، پیچیدگی زمانی را تا $O(m \times n^2)$ افزایش می‌دهد. یعنی برای هر کدام از m سطر ما، n^2 عملیات صورت می‌گیرد. (رفتن و بازگشتن از هر خانه)

2. **روش برنامه‌ریزی پویا:**

استفاده از روش DP فاز اول با اجتناب از انتخاب سلول‌های مسدود شده به عنوان همسایه‌های خانه‌ی مورد نظر.

پیچیدگی زمانی:

پیچیدگی زمانی $O(m \times n)$ باقی می‌ماند زیرا الگوریتم هر خانه جدول را یک بار پیمایش می‌کند و سلول‌های مسدودشده صرفا نادیده گرفته می‌شوند.

بررسی بهینگی:

- **روش حریصانه:**

روش حریصانه به دلیل استفاده از استراتژی انتخاب محلی بهینه، ممکن است از مسیر بهینه دور شود. در این روش، در هر مرحله تنها بهترین انتخاب ممکن از میان همسایگان فعلی انجام می‌شود، بدون بررسی تمامی گزینه‌ها در طول مسیر. این می‌تواند منجر به انتخاب‌های نادرست و در نتیجه مسیر نهایی غیر بهینه شود. این مشکل به ویژه در مسائلی که ابعاد ماتریس بزرگ‌تر است و تعداد زیادی سلول مسدود وجود دارد، مشهود است.

- **برنامه‌ریزی پویا:**

الگوریتم برنامه‌ریزی پویا به دلیل بررسی تمامی مسیرهای ممکن و محاسبه بهترین گزینه در هر مرحله، همواره بهترین مسیر را ارائه می‌دهد. این روش تمامی گزینه‌ها را بررسی و مقایسه می‌کند، بنابراین می‌تواند از ابتدا تا انتهای مسیر به طور دقیق‌ترین انتخاب‌ها را انجام دهد و همیشه جواب بهینه را پیدا کند. با اینکه پیچیدگی زمانی این روش ممکن است بیشتر از روش حریصانه باشد، اما در مسائل پیچیده و با تعداد زیادی گزینه، دقت بیشتری دارد.

تحلیل و بررسی یک مثال:

Example:

Input:

```
[
  [-1, 24, 62, 54, 48, 53, 0, 0, 89, 45],
  [-1, 88, 7, 53, 70, 58, -1, -1, 15, 99],
  [0, 77, 92, 63, 87, 49, -1, 0, 66, 8],
```

```
    [-1, 84, 100, 60, 95, 38, 0, 0, 42, 73],
    [0, 0, 16, 1, 93, 36, 0, 0, 70, 91],
    [-1, 62, 44, 16, 16, 20, 0, -1, 32, 77],
    [-1, 73, 28, 22, 90, 97, -1, -1, 54, 49],
    [0, 0, 65, 41, 99, 16, -1, 0, 88, 45],
    [-1, 57, 18, 76, 53, 87, -1, -1, 66, 49],
    [-1, 54, 49, 21, 41, 65, 0, -1, 47, 90],
]
```

Greedy Approach:

Max Happiness: 545

Path: [3, 2, 3, 3, 4, 3, 4, 4, 5, 5]

Time Complexity: O(m * n * n), m=10, n=10, 0.999 milliseconds

Memory Usage: 0.152 KB

DP Approach:

Max Happiness: 579

Path: [3, 4, 4, 3, 3, 2, 1, 1, 1, 2]

Time Complexity: O(m * n), m=10, n=10, 0.000 milliseconds

Memory Usage: 0.488 KB

فاز سوم:

در این فاز بنزین در کشور کریم گران شده. او برای صرفه‌جویی در مصرف بنزین، ترجیح میدهد حرکتهای اریب کمتری انجام دهد. حرکت از خانه ای به سمت راست آن خوشحالی کریم را یک واحد افزایش و انجام حرکت به صورت اریب خوشحالی کریم را یک واحد کاهش می‌دهد.

ورودی:

ورودی این فاز کاملا مانند فاز قبلی‌ست.

خروجی:

- حداکثر مقدار شادی قابل دستیابی با توجه به در نظر گرفتن تعداد حرکت‌های مستقیم و یا اریب
- مسیر طی‌شده با اجتناب از سلول‌های مسدود شده و در نظر گرفتن نقاط یخچال و همچنین در نظر گرفتن مسیری که خوشحالی را بیشینه می‌کند.

تفاوت‌های کلیدی این فاز:

در این فاز سیستم جریمه و پاداش در نظر گرفته می‌شود. بدین صورت که برای هر حرکت مستقیم شادی +1 شده و برای هر حرکت اریب شادی -1 می‌شود. این شرایط در انتخاب راه در هر دو روش حریصانه و برنامه ریزی پویا تاثیر می‌گذارد.

توضیح دو روش:

1. **روش حریصانه**

در این روش، مانند فاز دوم، ابتدا بهترین خانه را برای شروع انتخاب می‌کنیم. سپس با بررسی همسایه‌های خانه انتخاب‌شده، به ستون‌های بعدی حرکت می‌کنیم. مانند فاز دوم، در هر مرحله همیشه بیشترین ارزش ممکن برای همسایه‌ها را انتخاب می‌کنیم. با این حال، در این روش باید این نکته را در نظر بگیریم که اگر **همسایه در جهت اریب باشد، یک واحد از ارزشی که کسب می‌شود کم** می‌شود، در حالی که اگر همسایه در جهت **مستقیم باشد، یک واحد به ارزش افزوده** می‌شود. سپس با در نظر گرفتن این تغییرات، بیشترین ارزش را از بین همسایه‌ها انتخاب می‌کنیم و مسیر را ادامه می‌دهیم. اگر مسیر معتبری پیدا نشد، به ستون قبلی باز می‌گردیم و دوباره تلاش می‌کنیم.

پیچیدگی زمانی:

در این الگوریتم برای تک تک n سطر هر ستون، 3 همسایگی آن را چک می‌کنیم که $O(n)$ را به ما نتیجه می‌دهد. در کل m ستون داریم. بنابراین پیچیدگی زمانی کل الگوریتم $O(m \times n)$ می‌شود.

2. روش برنامه‌ریزی پویا

پیچیدگی زمانی:

برای هر ستون(m)

برای هر سطر(n)

بیشترین مقدار از بین سه همسایگی آن را انتخاب می‌کند. $O(1)$

بنابراین پیچیدگی زمانی کل $O(m \times n)$ می‌شود.

بررسی بهینگی:

در روش برنامه‌سازی پویا پیچیدگی حافظه(Space Complexity) در الگوریتم $O(m \times n)$ خواهد بود. در حالی که در حل مسئله به روش حریصانه، پیچیدگی حافظه $O(m)$ **خواهد بود** که این موضوع باعث می‌شود این الگوریتم برای ماتریس‌هایی با ابعاد بزرگ‌تر، از لحاظ مصرف حافظه بهینه‌تر باشد.

تحلیل و بررسی یک مثال:

Example:

Input:

```
[
  [-1, 24, 62, 54, 48, 53, 0, 0, 89, 45],
  [-1, 88, 7, 53, 70, 58, -1, -1, 15, 99],
  [0, 77, 92, 63, 87, 49, -1, 0, 66, 8],
  [-1, 84, 100, 60, 95, 38, 0, 0, 42, 73],
  [0, 0, 16, 1, 93, 36, 0, 0, 70, 91],
  [-1, 62, 44, 16, 16, 20, 0, -1, 32, 77],
  [-1, 73, 28, 22, 90, 97, -1, -1, 54, 49],
  [0, 0, 65, 41, 99, 16, -1, 0, 88, 45],
  [-1, 57, 18, 76, 53, 87, -1, -1, 66, 49],
```

```
    [-1, 54, 49, 21, 41, 65, 0, -1, 47, 90],
]
```

Greedy Approach:
Max Happiness: 545
Path: [3, 2, 3, 3, 4, 3, 4, 4, 5, 5]
Time Complexity: O(m * n), m=10, n=10, 0.103 milliseconds
Memory Usage: 0.152 KB

DP Approach:
Max Happiness: 579
Path: [3, 4, 4, 3, 3, 2, 1, 1, 1, 2]
Time Complexity: O(m * n), m=10, n=10, 0.358 milliseconds
Memory Usage: 0.492 KB

درس‌آموخته‌ها:

- تفاوت مصرف حافظه در روش‌های حریصانه و برنامه‌سازی پویا:

روش Greedy: در این روش، حافظه مصرفی به نسبت کم‌تر است، زیرا نیازی به ذخیره‌سازی اطلاعات برای تمام خانه‌ها نداریم. فقط وضعیت فعلی و همسایگان مربوطه بررسی می‌شوند.

روش DP: در این روش، به دلیل نیاز به ذخیره‌سازی مقادیر بیشینه در هر خانه از ماتریس، مصرف حافظه به مراتب بیشتر می‌شود. برای ماتریس‌های بزرگ، ممکن است این میزان حافظه مصرفی به مشکل تبدیل شود.

- تفاوت سرعت در ماتریس‌های کوچک و بزرگ:

ماتریس‌های کوچک: در ماتریس‌های کوچک، هر دو روش Greedy و DP به طور تقریبی عملکرد مشابهی دارند و تفاوت عملکرد قابل توجهی بین آن‌ها مشاهده نمی‌شود.

ماتریس‌های بزرگ: در ماتریس‌های بزرگ، روش Greedy به دلیل اینکه فقط همسایگان مستقیم خانه‌های فعلی را بررسی می‌کند و نیازی به ذخیره‌سازی همه‌ی مقادیر ندارد، سرعت بهتری خواهد داشت. اما روش DP با محاسبه تمامی گزینه‌ها و ذخیره‌سازی مقادیر، ممکن است زمان بیشتری ببرد، ولی همیشه مسیر بهینه را می‌یابد.

در نتیجه، اگر ماتریس بسیار بزرگ باشد و نیاز به استفاده از حافظه زیاد و زمان محاسباتی طولانی وجود داشته باشد، روش Greedy می‌تواند گزینه بهتری باشد. در حالی که برای اطمینان یافتن از مسیر بهینه، به خصوص در ماتریس‌های پیچیده‌تر یا ماتریس‌هایی با ویژگی‌های خاص، روش DP انتخاب بهتری است.