دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده ی ریاضی و علوم کامپیوتر

پایان‌نامه کارشناسی‌ارشد

گرایش علوم کامپیوتر

سرچ تخاصمی

گزارش چهارم

نگارش

پویاپارسا

استاد راهنما

دکتر قطعی

فروردین ۱۴۰۰


بسم الله الرحمن الرحيم

فصل اول

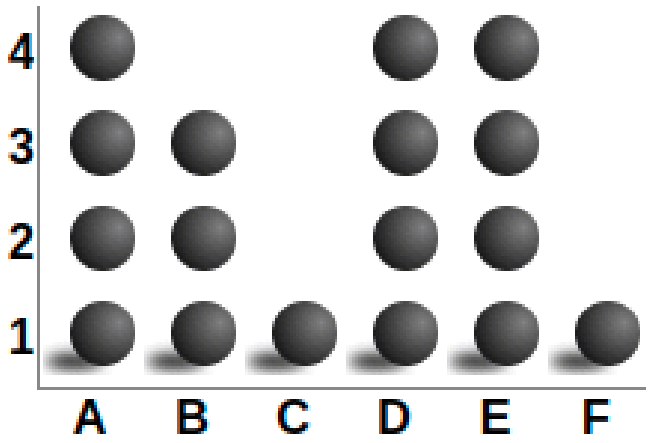
تعریف مسئله

۱-۱ مقدمه

جستجوی تخاصمی در موقعیت هایی به کار گرفته می شود که رقیب ای رو به روی شما وجود دارد که هر در قدم می خواهد در خلاف جهت منافع شما حرکت کند. این ویژگی دقیقا در بازی Nim وجود دارد رقیب شما سعی می کند اکشن هایی انجام دهد تا منجر به باخت شما بشود. [۲]

۱-۲ صورت مسئله

بازی Nim یک بازی ریاضیاتی استراتژیک است که در آن دو بازیکن به نوبت شروع به برداشتن قطعات می کنند هر بازیکن باید در هر نوبت ، حداقل یک قطعه بردارد و می تواند هر تعداد دلخواه بیشتر از یک را حذف کند به شرطی که از یک ستون باشند در انتها فردی بازنده است که آخرین قطعه را بردارد. برای اطلاعات بیشتر در مورد این بازی می توانید به [۳] رجوع کنید.



شکل ۱-۱: نمایی از بازی در شروع.

برای دیدن بازی در عمل می توانید به این لینک مراجعه کنید.
هر چند در نگاه اول این بازی ساده به نظر می رسد ولی پس از چند دقیقه بازی کردن متوجه تعداد زیاد احتمالات در این بازی خواهید شد. در واقع رشد گراف این مسئله شما رو شگفت زده خواهد کرد!

فصل دوم

مدل سازی

۲-۱ رویکرد

ابتدا به نحوه ی ساخت گراف اشاره می کنیم سپس به محدودیت هایی که باعث می شود ساخت کل گراف به در عمل ممکن نباشد می پردازیم و سعی می کنیم با استفاده از تکنیک هایی مانند هرس کردن با ساخت بخش کوچکی از گراف به حل مسئله بپردازیم.

۲-۲ ساخت گراف

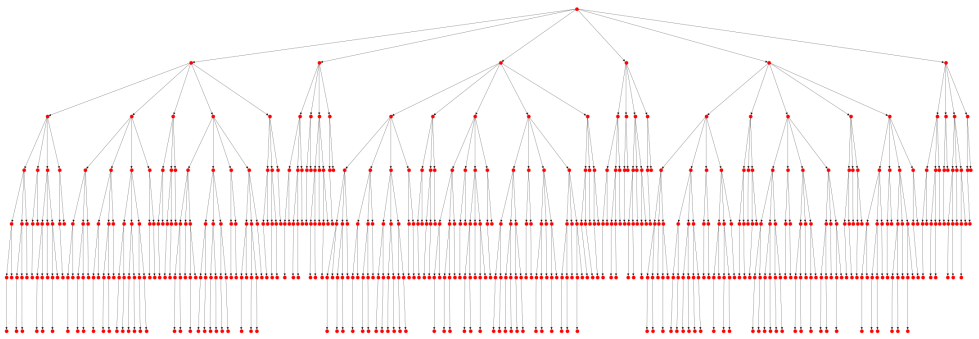
برای ساخت گراف مسئله باید به دو سوال جواب دهیم :

الف) رئوس، نماینده چه هستند؟

ب) چه زمان دو راس به هم متصل می شوند؟

وقتی که نوبت شما در بازی می شود شما باید دو مولفه را انتخاب کنید؛ از کدام ستون می خواهید انتخاب کنید و به چه تعداد؛ رئوس دقیقا نماینده پاسخ این دو سوال هستند: هر راس به شکل شماره ستون - تعداد انتخابی نامگذاری شده است؛ توجه کنید که شماره ستون ها از صفر آغاز می شود ولی تعداد قطعات انتخابی از یک.

اتصال راس ها به هم نیز نشان دهنده ی حرکت بازیکن هاست مثلا وقتی از راس 0-1-0 به راس 0-1-3-0-1، یالی وجود دارد به این معناست که بازیکن اول از ستون اول یک قطعه حذف کرده است و پس از آن بازیکن دوم از ستون دوم سه قطعه حذف کرده است.



شکل ۲-۱: گراف ساخته شده برای سه ستون دوتایی.

دقت کنید برای ساده ساختن پیاده سازی، گراف به طور جهت دار از سمت ریشه به سمت برگ ها تعریف شده است؛ با این تکنیک ساده همسایه های یک راس همان فرزندان آن هستند.

۲-۳ چرا نباید گراف را به طور کامل ساخت ؟

به دلیل ماهیت نمایی پیچیدگی مسئله که به طور تجربی آن را $O(۲۵^n)$ حدس می زنم و در آن n تعداد ستون هاست به راحتی می توان دریافت که رشد گراف به طرز خیره کننده ای سریع است.
برای درک این رشد کافی است به این ارقام دقت کنید :

۱. یک ستون چهارتایی کمتر از یک ثانیه

۲. دو ستون ۴ تایی نزدیک به ۳۰ ثانیه

۳. سه ستون ۴ تایی بیش از یک ساعت

۴. چهار ستون ۴ تایی نزدیک به یک روز

۵. پنج ستون ۴ تایی نزدیک به یک ماه طول خواهد کشید!

فصل سوم

پیاده سازی

۳-۱ مقدمه

ابتدا به نحوه پیمایش گراف و محاسبه ی امتیازات از روی درخت حالات پرداخته و سپس به پیاده سازی تکنیک هرس کردن می پردازیم.
در این پیاده سازی از کتاب خانه ی قدرتمند [۱] استفاده شده است.
کدهای این گزارش از طریق این لینک در دسترس است .

۳-۲ پیمایش در گراف

ابتدا اقدام به ساخت سطح به سطح درخت حالات کردم اما مشکل این کار اینست که نتیجه ی بازی تا انتهای آن مشخص نخواهد شد لذا برای ایده هایی مانند هرس کردن باید یک عمق را تا انتها رفته باشید تا بتواند تصمیم بگیرید که آیا به عمق های دیگر بروید یا خیر. لذا گراف همانطور که ساخته می شود پیمایش نیز می شود و مقادیر امتیازات محاسبه می شود دو حلقه ی تو در تو و بازگشتی صدا زدن تابع make graph پیمایش DFS در گراف را میسر کرده اند.

```
26    for pile_index, pile_stones in enumerate(game_states[parent_node]):
27
28      if game_states[parent_node][pile_index] <= 0 :
29        continue
30
31      for stone_to_take in range(1, pile_stones + 1):
32        node = str(pile_index) + "-" + str(stone_to_take) + "_" + parent_node
33        G.add_edge(parent_node, node)
34        labels[node] = node
35
36        game_state = game_states[parent_node].copy()
37        pile, stone = node.split("_")[0].split("-")
38        game_state[int(pile)] -= int(stone)
39        game_states[node] = game_state
40        make_graph(game_states, G, labels, node, values, scores, dead_nodes)
```

شکل ۳-۱: ساخت و پیمایش هم زمان گراف به روش DFS.

۳-۳ نحوه ی محاسبه ی امتیازات

همان طور که در ادامه خواهیم دید هرس کردن می تواند مسئله را در زمان قابل قبول حل کند به همین دلیل از Estimation استفاده ای نشده است و امتیازات به صورت صفر و یک که نشان دهنده باخت و برد نفر شروع کننده هستند درآمده است. همان طور که گفته شد باید به برگی برسیم تا بتوانیم بفهمیم که نتیجه حرکت در لایه های بالایی چه بوده است؛ از طرفی به علت به کارگیری تکنیک هرس کردن نمی توانیم کل امتیازات برای هر راس را محاسبه کنیم و سپس با یک پیماش امتیازات را برای هر راس

محاسبه کنیم در واقع باید این آمادگی را داشته باشیم که در هر لحظه پیمایش درخت تمام شود و لذا همواره باید درخت ما اعداد درست را در خود داشته باشد؛ برای تحقیق این امر تابع ای به نام update node معرفی کرده ام؛ ابتدا از طریق زوج و فرد بودن سطح برگ، می فهمیم که برنده ی بازی نفر اول بوده است یا دوم سپاس با استفاده از این تابع تمام نود هایی که ممکن است با مشخص شدن مقدار این برگ مقدارشان تغییر کند تا رسیدن به ریشه را به روز می کنیم.

```python
def update_node(G, node, values, scores, dead_nodes):

  while len(list(G.predecessors(node))) > 0:

    # node level starting from zero
    node_level = get_node_level(node)

    # parent node
    parent = list(G.predecessors(node))[0]

    values[parent][node] = scores[node]

    # parent is MAX player
    if (node_level - 1) % 2 == 1:
      scores[parent] = max(list(values[parent].values()))
      if scores[parent] == 1:
        dead_nodes.append(parent)

    # parent is MIN player
    elif (node_level - 1) % 2 == 0:
      scores[parent] = min(list(values[parent].values()))
      if scores[parent] == 0:
        dead_nodes.append(parent)

    node = parent

```

شکل ۳-۲: تخمین مدل از بهبود در آینده.

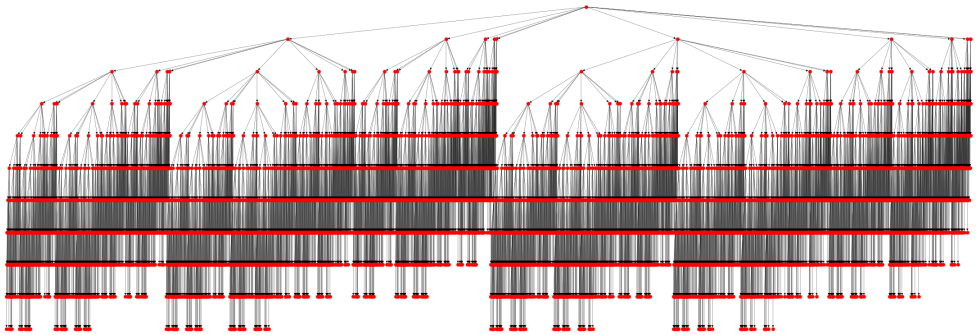
۳-۴ هرس کردن :

به جرات می توانم بگویم جذاب ترین بخش این گزارش برای من این بخش است زیرا با این ایده ی ساده و زیبای هرس کردن بخش بزرگی از پیچیدگی مسئله در هم شکسته می شود.
همان طور که می دانیم تکنیک هرس کردن زمانی بیش ترین اثر را داراست که مسئله well-order باشد یعنی موارد که کران های مناسب را معرفی می کنند ابتدا ظاهر شوند هر چند بازی Nim این ویژگی را به طور کامل دارا نیست ولی مثلا در حالتی که بازی با دو ستون پنج تایی شروع می شود ریشه دارای

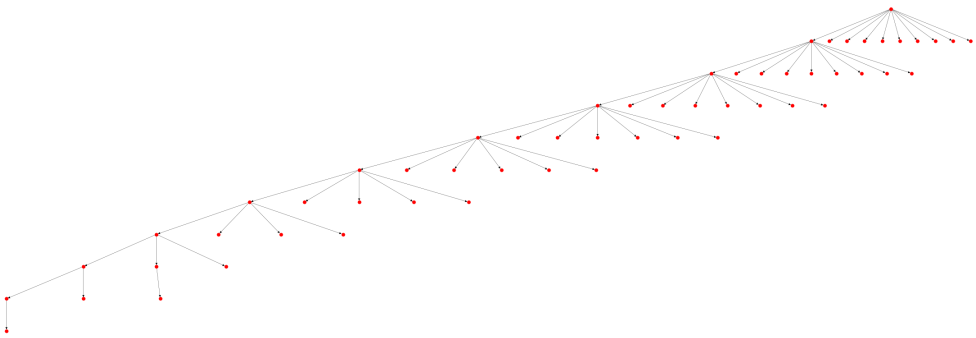
ده فرزند است که با پیمایش اولین فرزند جواب قطعی مسئله روشن می شود و علاوه بر آن در حین پیمایش اجزای همین فرزند نیز هرس کردن استفاده می شود و از این طریق زمان اجرای برنامه از پنج دقیقه و چهل ثانیه به سه ثانیه تقلیل می یابد !

```
3   # alpha beta pruning
4   if parent_node in G.nodes():
5     parent_list = list(G.predecessors(parent_node))
6
7     if len(parent_list) > 0:
8       parent = list(G.predecessors(parent_node))[0]
9
10      if parent in dead_nodes:
11        return
12
```

شکل ۳-۳: پیاده سازی هرس کردن در make graph.



شکل ۳-۴: کل گراف حالات.



شکل ۳-۵: گراف هرس شده.

هرس کردن نیز بسیار ساده انجام می گیرد، سطر های درخت ما به صورت یکی در میان به دنبال ماکزیمم و مینمم هستند لذا اگر در فرندان یک نود ماکزیمم یک ظاهر شد دیگر نیازی به چک کردن

سایر فرندان نیست و همچنین اگر در فرزندان یک نود مینیمم، صفر ظاهر شد دیگر فرزندان نیاز به بررسی ندارند.

فصل چهارم

تحلیل حساسیت

در این بخش به مقایسه زمان اجرای حالت کل گراف و گراف با استفاده از هرس کردن می پردازیم همچنین سایز ورودی نیز تغییر داده می شود :

تعداد ستون ۴ تایی	کامل	هرس شده
۲	۳۰ ثانیه	۱ ثانیه
۳	بیش از یک ساعت	۳ ثانیه
۴	بیش از یک روز	۶ ثانیه

جدول ۴-۱: زمان اجرا با سایز ورودی متفاوت

تمایز بین استفاده و عدم استفاده از هرس کردن نیز اعجاب انگیز است.

منابع و مراجع

[1] networkx. networkx. https://networkx.org.

[2] Pomona, Cal Poly. Adversarial search. https://www.cpp.edu/~ftang/courses/CS420/notes/adversarial%20search.pdf.

[3] Wikipedia. Mnim. https://en.wikipedia.org/wiki/Nim#:~:text=Nim%20is%20a%20mathematical%20game,from%20distinct%20heaps%20or%20piles.&text=This%20is%20called%20normal%20play,way%20that%20Nim%20is%20played.