

توضیحات پروژه csp

آریا ابراهیمی ۹۸۲۲۷۶۲۱۷۵

مدل سازی

- متغیر ها را خانه های جدول هستند بنابراین برای یک صفحه $B_{n \times m}$ ، $n \times m$ متغیر خواهیم داشت. از آنجایی که هر کدام از متغیر ها یک قطب آهنربا هستند، یک تابع به نام `get_other_pole` در نظر گرفته شده که با دریافت مختصات یک قطب، مختصات قطب دیگرش را برمیگرداند و هنگام مقدار دهی این دو قطب را با یکدیگر مقدار میدهیم و در دامنه هایشان هم با یکدیگر آپدیت میشود. میتوانستیم پیاده سازی را کلاً به این صورت در نظر بگیریم که

هر دو قطب یک متغیر شوند که در این صورت $\frac{n \times m}{2}$ متغیر حاصل میشود.

- دامنه مقادیری است که هر متغیر میتواند اختیار کند که در این مسئله، مقادیر مثبت، منفی و بی مقدار هستند که در کد به صورت `'+', '-', ''` نمایش داده شده اند. همانطور که در قسمت قبل هم توضیح داده شد، اگر متغیری مقدار `+` بگیرد با استفاده از تابع `get_other_pole` مختصات قطب دیگرش گرفته می شود و مقدار `-` برای آن اختیار میشود.

توضیحات کد

- در قسمت `main` ورودی ها از فایل دریافت می شوند و تابع `backtrack` صدا زده میشود.
- در فایل `csp.py` توابع مربوطه پیاده سازی شده اند که به توضیح هر کدام میپردازیم.
- `Backtrack`: الگوریتم اصلی است که همانند شبه کد زیر عمل میکند:

CSP-BACKTRACKING(A)

1. If assignment A is complete then return A
2. $X \leftarrow$ select a variable not in A
3. $D \leftarrow$ select an ordering on the domain of X
4. For each value v in D do
 - a. Add $(X \leftarrow v)$ to A
 - b. If A is valid then
 - i. $result \leftarrow$ CSP-BACKTRACKING(A)
 - ii. If $result \neq$ failure then return $result$
 - c. Remove $(X \leftarrow v)$ from A
5. Return failure

شکل ۱: شبه کد `backtrack`

که برای انتخاب متغیر (X در شبه کد) از الگوریتم MRV استفاده میکنیم و برای ترتیب انتخاب دامنه (D در شبه کد) از الگوریتم LCV استفاده میکنیم.

MRV: روی متغیرها پیمایش میکند و متغیری که هنوز مقداردهی نشده است و کمترین دامنه باقی مانده را دارد برمیگرداند.
LCV: روی دامنه متغیر انتخاب شده پیمایش میکند و دامنه را بر اساس محدودیت‌هایی که بین همسایه‌ها و متغیر فعلی و قطب دیگرش ایجاد میشود، به صورت صعودی سورت میکند.

- **is_safe**: محدودیت‌ها را بررسی میکند، در این مسأله دو سری محدودیت داریم
 ۱. محدودیت‌های دوتایی بین متغیرها (قطب‌های همنام نمیتوانند کنار هم باشند)
 ۲. محدودیت‌های سراسری که تعداد بیشتری از متغیرها را درگیر میکند (برای مثال یک سطر بیشتر از ۲ قطب مثبت نداشته باشد).

الگوریتم ارائه شده در **is_safe**، این محدودیت‌ها را بررسی میکند در ابتدا چک میکند که دو متغیر کنار هم مقدار یکسانی نداشته باشند و اگر چنین بود مقدار False برمیگرداند. در ادامه چک میکند که اگر تمامی متغیرهای یک سطر یا ستون مقداردهی شده بودند ولی محدودیت مربوط به آن ارضاء نشده بود همانجا False برمیگرداند و در غیر این صورت (متغیرهای سطر یا ستون کامل مقداردهی نشده بودند) چک میکند اگر تعداد مقادیر داده شده برای متغیرها، محدودیت‌ها را رد کرد بازهم False برمیگرداند و در آخر اگر هیچکدام از این اتفاق‌ها نیافتاده بود، True برمیگرداند.

- **Forward Check**: الگوریتم forward check روی همسایه‌های متغیر مقداردهی شده پیمایش میکند و مقادیری در دامنه آن‌ها که باعث ایجاد محدودیت می‌شود را حذف میکند. همچنین چک میکند که اگر دامنه یکی از همسایه‌ها تهی شد، مقدار False ریترن میکند زیرا آن متغیر دیگر نمیتواند مقداری اختیار کند. البته در این مسأله متغیرها میتوانند مقدار پوچ را در هر مرحله اختیار کنند بنابراین این اتفاق که دامنه متغیر تهی شود اتفاق نمی‌افتد.
- **Ac3**: الگوریتم ac3 همانند شبه کد زیر کار میکند و ورژن بزرگتری از forward check می‌باشد و در محدوده بزرگتری دامنه‌ها را تغییر میدهد. به دلیل زمان بر بودن این الگوریتم فقط چندبار صدا زده می‌شود (یکبار در ابتدای اجرا و یکبار در وسط اجرای برنامه).

AC3

1. **contradiction** \leftarrow false
2. Initialize queue **Q** with all variables (not yet instantiated)
3. While **Q** $\neq \emptyset$ and \neg **contradiction** do
 - a. **X** \leftarrow Remove(**Q**)
 - b. For every (not yet instantiated) variable **Y** related to **X** by a (binary) constraint do
 - If REMOVE-VALUES(**X**,**Y**) then
 - i. If **Y**'s domain = \emptyset then **contradiction** \leftarrow true
 - ii. Insert(**Y**,**Q**)

شکل ۲: الگوریتم ac3

REMOVE-VALUES(**X**,**Y**)

1. **removed** \leftarrow false
2. For every value **v** in the domain of **Y** do
 - If there is no value **u** in the domain of **X** such that the constraint on (**X**,**Y**) is satisfied then
 - a. Remove **v** from **Y**'s domain
 - b. **removed** \leftarrow true
3. Return **removed**

شکل ۳: الگوریتم revise

```

-----
- + - + - +
+ - + - + - + -
- + - + - + - +
+ + - + - + -
- -
+ - - + +
- + - + - +
+ - + - + -
- + - + - + -
- + - + - + -
-----
total time: 0.2914543151855469

```

شکل ۴: جواب با استفاده از *ac3*

```

-----
- + - + - +
+ - + - + - + -
- + - + - + - +
+ + - + - + -
- -
+ - - + +
- + - + - +
+ - + - + -
- + - + - + -
- + - + - + -
-----
total time: 0.30308985710144043

```

شکل ۵: جواب بدون *ac3*

```

-----
- +
+ - + -
+ + - +
- + - -
+ - +
- - +
-----
total time: 0.03631925582885742

```

شکل ۶: جواب تست اول با استفاده از *ac3*

با استفاده از الگوریتم *ac3* دامنه متغیرها را کاهش میدهیم و در نتیجه حالات کلی که نیاز به بررسی است تا به جواب برسیم کمتر می شود و همچنین در زمان پاسخ، تأثیر چندانی ندارد.