

木幅アルゴリズムの学習システムの構築

データベース連携の修正と画像生成・サーバー選び

B4 小林紹子

November 11, 2025

目次

- 1 データベース連携の修正（prisma の設定ファイルの変更）
- 2 React Flow における画像生成の修正

目次

1 データベース連携の修正（prisma の設定ファイルの変更）

2 React Flow における画像生成の修正

WSL 環境でのエラー

- なぜか Windows の VScode (WSL) で Web アプリケーションにエラーが出た.
- データベースと Next.js の統合がうまく行かない.
- .env ファイルにある環境変数が読み込まれないことが原因.
- prisma.config.ts に `import 'dotenv/config'` を追加することで解消できた.

エラーの原因候補

- 1 Windows 上の VSCode + WSL 環境では、環境変数の伝播や改行コードの違いにより、`.env` が正しく読まれないことがある。
- 2 Prisma CLI は内部で `dotenv` を呼び出すが、WSL 経由では `process.env` に値が渡らない場合がある。
- 3 Ubuntu ネイティブ環境では、シェル経由で正しく環境変数が設定されるため問題が起きなかった。

prisma.config.ts と環境変数の自動読み込み

- Prisma 5 まででは、CLI が自動的に `.env` を読み込んでいた.
- Prisma 6 以降では、設定を TypeScript モジュールとして記述する `prisma.config.ts` が導入された.
- 設定ファイルが TypeScript/ESM モジュールとして扱われる.
- 仕様変更: 「TypeScript モジュールとして読み込まれる設定ファイルからは、`.env` を自動で読み込まない」
- 理由: ESM モジュール読み込み時に環境変数を暗黙的に扱うと挙動が不明確になるため.
- そのため、明示的に `import "dotenv/config";` を追加して環境変数を読み込む必要がある.

Prisma CLI とは

- Prisma の各種操作をターミナルから実行するためのコマンド群.
- データベーススキーマの管理やクライアント生成を行う.
- 例：
 - `npx prisma init` : 初期設定
 - `npx prisma generate` : Prisma Client 生成
 - `npx prisma migrate dev` : DB マイグレーション
 - `npx prisma studio` : GUI でデータ確認
- CLI は内部で `.env` を読み込んで環境変数を設定する.
- ただし WSL 環境ではこの自動読み込みが失敗する場合がある.

明示的に .env を読み込む (prisma.config.ts)

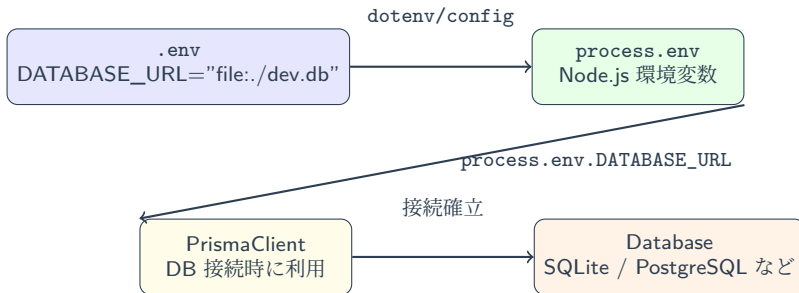
```
1 import "dotenv/config";  
2 import { PrismaClient } from "@prisma/client";  
3  
4 const prisma = new PrismaClient();  
5 export default prisma;
```

- この1行により、Node.js 起動時に .env が読み込まれる。
- PrismaClient が process.env.DATABASE_URL を利用可能に。

環境変数と `process.env`

- Node.js にはグローバルオブジェクト `process` が存在する.
- その中の `process.env` は, OS や `.env` ファイルに定義された環境変数を保持.
- 例: `process.env.DATABASE_URL` はデータベース接続 URL を表す.
- `dotenv/config` を読み込むことで, `.env` の内容が `process.env` に展開される.

環境変数の流れ (Prisma 利用時)



- `dotenv/config` が `.env` の内容を `process.env` に展開.
- `PrismaClient` が `process.env.DATABASE_URL` を利用して DB に接続.

目次

1 データベース連携の修正（prisma の設定ファイルの変更）

2 React Flow における画像生成の修正

前回までの課題点と解決策

課題点:

- 頂点から出る辺の位置が固定されている.
- 辺同士がぶつかり, 綺麗なグラフになっていない.

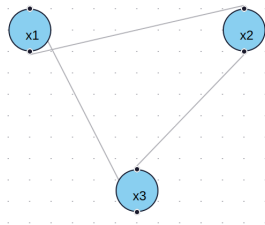


Figure: 今までの生成図形

解決策:

⇒ 辺を出す位置を頂点の中心の裏側から出るようにする.

React Flow の全体構造

React Flow とは

React Flow は「Node (頂点)」と「Edge (線)」で構成されたグラフを、ブラウザ上で直感的に表示・操作できるライブラリ。

- 主な構成要素：
 - **Node (ノード)**：丸や四角などの図形。
 - **Edge (エッジ)**：ノード同士をつなぐ線。
 - **Handle (ハンドル)**：ノード上の「線の接続点」。
 - **ReactFlowProvider**：全体の状態（ノード・エッジ・選択状態など）を管理。
- ノードやエッジの配置・ズームなどは React Flow が自動的に処理。
- 独自の見た目を作るときは CustomNode のようにコンポーネントを定義。

各構成要素のプロパティ

- **Node :**

- id : 一意な識別子.
- type : 表示に使うコンポーネントの種類 (例: CustomNode) .
- position : 座標 (x, y) .
- data : ノードに渡すカスタムデータ (例: ラベル名など) .

- **Edge :**

- source : 出発ノードの id.
- target : 到着ノードの id.
- type : エッジの種類 (直線・曲線など) .

各構成要素のプロパティ II

- **Handle :**
 - type : "source" または "target" (エッジの出発点／到達点を指定) .
 - id : ハンドルの識別子 (ノード内で複数ハンドルを区別) .
 - position : ハンドルの位置 (上・下・左・右) を Position.Top / Bottom / Left / Right で指定.
 - style : CSS で見た目をカスタマイズ (例: 透明にして中心に配置) .
 - isConnectable : 接続を許可するかどうか (true/false) .
- **ReactFlow コンポーネント**がこれらを統合し, 描画とイベント処理を行う.

Node Component の作成 I

Node Component (ノードコンポーネント) :

「ノード単体の見た目と挙動 (色・形・サイズ・ラベル・Handle の配置など)」を定義する場所.

```
1 export default function CustomNode({ data, selected }: NodeProps)
  {
2   return (
3     <div
4       style={{
5         background: selected ? "#FFD700" : "#89CFF0",
6         borderRadius: "50%",
7         width: 40,
8         height: 40,
9         display: "flex",
10        justifyContent: "center",
```


Node Component の作成 II

```
11         alignItems: "center",
12         fontWeight: "bold",
13         color: "#000",
14         position: "relative",
15     }}
16 >
17     {data.label}
18
19     <Handle
20         type="source"
21         id="center"
22         position={Position.Right}
23         style={{ ... }}
24     />
25
26     <Handle
```

Node Component の作成 III

```
27         type="target"  
28         id="center"  
29         position={Position.Left}  
30         style={{ ... }}  
31     />  
32 </div>  
33 )  
34 }
```

CustomNode の基本構造

```
1 export default function CustomNode(  
2   { data, selected }: NodeProps  
3 ) {  
4   return (  
5     <div>  
6       {data.label}  
7       <Handle type="source" id="center"  
8         />  
9       <Handle type="target" id="center"  
10        />  
11     </div>  
12   )  
13 }
```

- **CustomNode:**
React Flow における「ノード 1 つの描画定義」を担うコンポーネント.
- **NodeProps:** React Flow が各ノードに渡す情報の型.
- 引数 **data, selected:**
 - **data.label:** ノードのラベル.
 - **selected:** ユーザーが選択中かどうか.
- **<Handle> :**
エッジ接続点. 1 つのノード内に複数設置できる.

ノード本体のスタイル

```
1 <div
2   style={{
3     background: selected ? "#FFD700" :
4       "#89CFF0",
5     borderRadius: "50%",
6     width: 40,
7     height: 40,
8     display: "flex",
9     justifyContent: "center",
10    alignItems: "center",
11    fontWeight: "bold",
12    color: "#000",
13    position: "relative",
14  }}
15  >
16    {data.label}
17 </div>
```

- background :
 - 選択時：金色（#FFD700）.
 - 非選択時：水色（#89CFF0）.
- borderRadius: "50%"：円形ノード.
- width × height : 40px × 40px.
- display: flex と中央揃えでラベルを中央配置.
- position: "relative" :
内部のハンドルを相対的に配置できるようにする.

Handle の役割 I

```
<Handle
  type="source"
  id="center"
  position={Position.Right}
  style={{
    top: "50%",
    left: "50%",
    transform: "translate(-50%, -50%)",
    background: "transparent",
    border: "none",
  }}
/>
```

- type:
"source" (線を出す) or "target" (線を受ける) .
- position:
接続位置 (例: Position.Left, Position.Right) .
- id:
エッジの sourceHandle / targetHandle と対応.
- style: CSS で位置や透明化を指定.
- この例ではどちらの Handle もノード中央に配置.
- 背景を透明にし、見えないが線を接続できる.

まとめ：React Flow と CustomNode の関係

まとめ

- React Flow は「ノードとエッジ」で構成されるグラフ描画ライブラリ。
- ノードの見た目を自由に作りたいとき、CustomNode を定義する。
- Handle によりノード同士を接続可能。
- 状態管理（選択・移動・接続）は React Flow が自動で行う。