UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Proiect IA

Generator de bracket-uri

Autori,
Costache Andrei, 1408A
Butnaru Dan, 1408A
Romașcu Ștefan, 1408A

IAȘI

2024

Cuprins

7 Teste și rezultate

7.1 Descrierea testelor efectuate

7.2 Analiza rezultatelor obținute

8 Concluzii și direcții viitoare

8.1 Concluzii generale

8.2 Direcții de extindere și îmbunătățire

Capitolul 1: Enunțarea temei

1.1 Contextul utilizării bracket-urilor

Bracket-urile sunt utilizate pe scară largă în competițiile sportive și e-sports pentru a organiza meciurile într-un format logic. Acestea permit participanților și organizatorilor să vizualizeze parcursul competiției și să stabilească reguli clare pentru avansarea în etapele următoare.

1.2 Scopul și importanța proiectului

Proiectul nostru își propune să dezvolte o aplicație care să genereze automat bracket-uri valide, respectând constrângerile legate de divizii și coeficienți de performanță. Aceasta oferă o soluție rapidă și eficientă pentru organizarea turneelor.

Capitolul 2: Arhitectura generală

2.1 Prezentarea componentelor aplicației

Aplicația include următoarele componente principale:

- **Interfața grafică** (GUI): Permite introducerea datelor referitoare la echipe (nume, divizie, coeficient).
- **Algoritm de generare**: Un modul care aplică metoda de forward-checking pentru a crea perechi valide de echipe.
- **Vizualizarea bracket-ului**: Componentă care afișează structura turneului într-un format grafic.

2.2 Relația dintre componente

Interfața grafică colectează datele utilizatorului, care sunt procesate de algoritmul de generare. Rezultatul este apoi transmis modulului de vizualizare pentru afișarea finală.

2.3 Algoritmul Forward Checking

Forward checking este o tehnică utilizată în rezolvarea problemelor de tip **CSP** (Constraint Satisfaction Problems), cum ar fi Sudoku, rezolvarea n-reginelor sau, în acest caz, generarea perechilor de echipe. Această tehnică:

1. **Prevină conflictele viitoare**:
    - La fiecare pas, elimină din lista de opțiuni acele valori care ar încălca constrângerile dacă ar fi selectate.
2. **Reduce spațiul de căutare**:
    - Prin verificarea constrângerilor înainte de a trece la următorul pas, forward checking reduce numărul de soluții candidate, făcând procesul mai eficient.
3. **Funcționare în acest cod**:
    - În funcția **forward_checking**, perechile care nu respectă constrângerile sunt excluse imediat.
    - Doar perechile valide sunt adăugate în soluție, reducând astfel numărul de cazuri explorate. Dacă se detectează un conflict (nu există echipe rămase care să poată forma perechi valide), funcția revine la pasul anterior și încearcă alte combinații. Acest proces este o combinație între **forward checking** și **backtracking**.

```
def calculate_matchups(self, teams):
    try:
        max_diff = int(self.max_difference.get().strip())
    except ValueError:
        messagebox.showerror("Invalid Input", "Max Coefficient Difference must be an integer.")
        return []

    def is_valid_matchup(team1, team2):
        if not self.allow_same_division.get():
            return team1.division != team2.division and abs(team1.coefficient - team2.coefficient) <= max_diff
        return abs(team1.coefficient - team2.coefficient) <= max_diff

    def forward_checking(remaining_teams, matchups):
        if len(matchups) == len(teams) // 2:
            return matchups

        for i in range(len(remaining_teams)):
            for j in range(i + 1, len(remaining_teams)):
                team1, team2 = remaining_teams[i], remaining_teams[j]
                if is_valid_matchup(team1, team2):
                    matchups.append((teams.index(team1), teams.index(team2)))
                    next_remaining = [t for k, t in enumerate(remaining_teams) if k not in (i, j)]
                    result = forward_checking(next_remaining, matchups)
                    if result:
                        return result
                    matchups.pop()
        return None

    matchups = forward_checking(teams, [])
    if not matchups:
        raise ValueError("No valid matchups found with the given constraints.")
    return matchups
```

Funcția **calculate_matchups** este responsabilă pentru generarea perechilor de echipe (matchups) care respectă anumite constrângeri. Codul se bazează pe forward checking pentru a verifica dacă o pereche este validă și pentru a genera toate perechile posibile.

Pași principali:

1. **Citirea diferenței maxime între coeficienți**:
    - Se citește valoarea maximă permisă a diferenței dintre coeficienții echipelor din interfață.
    - Dacă valoarea introdusă nu este validă (de exemplu, nu este un număr întreg), se afișează un mesaj de eroare și funcția returnează o listă goală.
2. **Funcția is_valid_matchup**:
    - Verifică dacă două echipe pot forma o pereche validă:
        - Dacă opțiunea "Allow same-division matchups" nu este activată, echipele trebuie să fie din divizii diferite.
        - În toate cazurile, diferența coeficienților dintre echipe trebuie să fie mai mică sau egală cu valoarea maximă specificată.
3. **Funcția forward checking**:
    - Este o funcție recursivă care încearcă să genereze toate perechile posibile respectând constrângerile impuse.
    - Dacă numărul perechilor generate este suficient pentru a acoperi toate echipele (jumătate din numărul total, deoarece fiecare pereche conține două echipe), funcția returnează soluția curentă.
    - Parcurge toate combinațiile posibile de echipe:
        - Verifică dacă perechea curentă este validă folosind **is_valid_matchup**.
        - Dacă perechea este validă, adaugă perechea în soluția curentă și continuă procesul cu echipele rămase.
        - Dacă nu se găsesc soluții valide mai departe, elimină ultima pereche adăugată (backtracking) și continuă să caute alte combinații.
4. **Generarea și verificarea soluției**:
    - Se apelează funcția **forward_checking** pentru a începe generarea perechilor.
    - Dacă nu se pot genera perechi valide, funcția aruncă o eroare.

2.4 Vizualizarea grafică a bracket-ului

Funcția display_bracket creează o fereastră nouă în care afișează vizual un arbore al turneului (bracket-ul) generat. Echipele sunt plasate la nivelurile inițiale, iar conexiunile dintre ele sunt desenate pentru a reprezenta meciurile, avansând până la finala turneului. Pozițiile și liniile sunt calculate dinamic pentru un layout clar și organizat.

```
def display_bracket(self, bracket):
    bracket_window = tk.Toplevel(self.root)
    bracket_window.title("Tournament Bracket")

    canvas = tk.Canvas(bracket_window, width=800, height=600, bg="white")
    canvas.pack()

    spacing = 50
    line_length = 100
    start_x = 50
    start_y = 20

    positions = []
    for i, (team1, team2) in enumerate(bracket):
        y1 = start_y + i * spacing * 2
        y2 = y1 + spacing

        canvas.create_text(start_x, y1, text=team1.get_name(), anchor="w", font=("Helvetica", 10))
        canvas.create_text(start_x, y2, text=team2.get_name(), anchor="w", font=("Helvetica", 10))

        canvas.create_line(start_x + 50, y1, start_x + 50 + line_length, y1)
        canvas.create_line(start_x + 50, y2, start_x + 50 + line_length, y2)
        canvas.create_line(start_x + 50 + line_length, y1, start_x + 50 + line_length, y2)

        mid_y = (y1 + y2) // 2
        positions.append(mid_y)
```

Această secțiune a funcției display_bracket iterează prin perechile de echipe din bracket-ul turneului, desenând numele echipelor și liniile care le conectează pentru a reprezenta vizual meciurile din prima rundă.

```
positions = []
for i, (team1, team2) in enumerate(bracket):
    y1 = start_y + i * spacing * 2
    y2 = y1 + spacing

    canvas.create_text(start_x, y1, text=team1.get_name(), anchor="w", font=("Helvetica", 10))
    canvas.create_text(start_x, y2, text=team2.get_name(), anchor="w", font=("Helvetica", 10))

    canvas.create_line(start_x + 50, y1, start_x + 50 + line_length, y1)
    canvas.create_line(start_x + 50, y2, start_x + 50 + line_length, y2)
    canvas.create_line(start_x + 50 + line_length, y1, start_x + 50 + line_length, y2)

    mid_y = (y1 + y2) // 2
    positions.append(mid_y)

current_positions = positions
current_start_x = start_x + 50 + line_length
while len(current_positions) > 1:
    next_positions = []
    for i in range(0, len(current_positions), 2):
        y1 = current_positions[i]
        y2 = current_positions[i + 1]
        mid_y = (y1 + y2) // 2

        canvas.create_line(current_start_x, y1, current_start_x + line_length, y1)
        canvas.create_line(current_start_x, y2, current_start_x + line_length, y2)
        canvas.create_line(current_start_x + line_length, y1, current_start_x + line_length, y2)

        next_positions.append(mid_y)

    current_positions = next_positions
    current_start_x += line_length

if current_positions:
    canvas.create_line(current_start_x, current_positions[0], current_start_x + line_length, current_positions[0])
```

Codul utilizează o buclă while pentru a itera prin pozițiile echipelor din runda curentă, construind niveluri succesive ale bracket-ului până la determinarea câștigătorului final.

1. Calculul pozițiilor intermediare

Pentru fiecare pereche de poziții **(y1, y2)** din runda curentă, se calculează poziția mediană **(mid_y = (y1 + y2) // 2**), care reprezintă punctul central unde se întâlnesc liniile de conexiune.

2. Trasarea liniilor

Linii orizontale sunt desenate de la fiecare poziție din runda curentă către punctul median.

O linie verticală este desenată între aceste două poziții la punctul median.

3. Actualizarea pozițiilor

Poziția mediană calculată este adăugată în lista next_positions, care reprezintă pozițiile echipelor pentru runda următoare, după procesarea tuturor perechilor din runda curentă.

4. Progresie

Coordonata *current_start_x* este incrementată cu *line_length* pentru a avansa poziția orizontală a liniilor în următorul nivel al bracket-ului.

Capitolul 3: Funcționalitatea

3.1 Introducerea datelor utilizatorului

Utilizatorul poate adăuga echipe specificând:

- Numele echipei.
- Divizia din care face parte.
- Coeficientul de performanță.

3.2 Generarea automată a bracket-urilor

Bracket-urile sunt generate respectând următoarele reguli:

- Echipele din aceeași pereche trebuie să fie din divizii diferite.
- Diferența coeficienților între echipele unei perechi nu trebuie să depășească 20.
- Numărul echipelor trebuie să fie o putere a lui 2.

3.3 Vizualizarea și interpretarea rezultatelor

Rezultatul este afișat sub forma unui bracket grafic, ușor de interpretat, oferind utilizatorului o privire clară asupra perechilor generate și progresului turneului.

Capitolul 4: Rolul fiecărui membru al echipei

- **Romașcu Ștefan:** Generarea bracket-ului, buton pentru editare și ștergere a unei echipe introduse, documentație.
- **Butnaru Dan:** Algoritmul de forward checking, modificarea parametrilor utilizați în forward checking din interfață, documentație.
- **Costache Andrei:** Testare, încărcare de date din fișiere, documentație, analiza complexității.

Capitolul 5: Complexitatea algoritmului

5.1 Caz favorabil

Dacă toate echipele respectă constrângerile și sunt sortate în prealabil, complexitatea algoritmului este **O(n log n)**, datorită operației de sortare.

5.2 Caz mediu

În situația tipică, unele perechi necesită ajustări, dar constrângerile sunt satisfăcute parțial. Complexitatea devine **O(n log n + k * n)**, unde **k** reprezintă numărul de conflicte rezolvate.

5.3 Caz defavorabil

Dacă multe echipe au coeficienți incompatibili sau fac parte din aceeași divizie, algoritmul poate explora combinații multiple, ducând la o complexitate de **O(n^2)**.

Capitolul 6: Corectitudinea algoritmului

6.1 Metodologia de verificare

Corectitudinea algoritmului este asigurată prin:

- Aplicarea sortării inițiale pentru reducerea combinațiilor invalide.
- Verificarea tuturor perechilor posibile utilizând forward-checking.

6.2 Validarea rezultatelor prin teste

Rezultatele algoritmului au fost validate prin teste automate, confirmând respectarea constrângerilor și generarea corectă a bracket-urilor.

Capitolul 7: Teste și rezultate

7.1 Descrierea testelor efectuate

Testarea aplicației s-a realizat utilizând biblioteca `unittest` din Python. Au fost dezvoltate teste automate pentru a verifica funcționalitatea aplicației, corectitudinea algoritmului și comportamentul acesteia în scenarii limită. Testele efectuate includ:

1. **Validarea introducerii datelor**
    - S-au verificat cazurile în care utilizatorul introduce date valide (nume echipă, divizie, coeficient numeric pozitiv). Aplicația a adăugat corect echipele în lista internă.
    - Pentru introducerea unor date invalide, cum ar fi coeficienți nenumerici sau lipsa unor câmpuri, aplicația a respins datele și a afișat mesaje de eroare adecvate.
2. **Generarea bracket-urilor**
    - S-a testat generarea bracket-urilor pentru scenarii valide, în care toate echipele respectă constrângerile (număr de echipe putere a lui 2, diferența maximă de coeficienți între echipele unei perechi este 20, iar echipele din pereche sunt din divizii diferite).
    - Pentru scenarii invalide, cum ar fi un număr insuficient de echipe sau număr de echipe care nu este o putere a lui 2, aplicația a afișat mesaje de eroare și nu a generat bracket-uri.
3. **Validarea algoritmului**
    - Algoritmul de generare a fost verificat în condiții optime (toate echipele respectă constrângerile) și în condiții limită, în care sunt necesare ajustări pentru a respecta regulile.
    - În cazurile în care nu se puteau forma perechi valide din cauza constrângerilor (toate echipele fiind în aceeași divizie sau având coeficienți incompatibili), aplicația a raportat corect imposibilitatea generării bracket-urilor.
4. **Testare la scară mare**
    - Aplicația a fost testată cu seturi mari de date (ex.: 64 de echipe). Rezultatele au arătat că aplicația funcționează eficient, generând perechi valide în timp util, chiar și pentru un număr mare de echipe.

7.2 Analiza rezultatelor obținute

Rezultatele au arătat că algoritmul funcționează conform așteptărilor, generând bracket-uri valide în toate cazurile testate. Capturile de ecran cu bracket-urile generate demonstrează respectarea regulilor impuse.

Capitolul 8: Concluzii și direcții viitoare

8.1 Concluzii generale

- Proiectul a îndeplinit obiectivele stabilite, oferind o aplicație intuitivă și funcțională pentru generarea de bracket-uri valide.
- Algoritmul utilizat este eficient și robust pentru seturi mici și medii de echipe.

8.2 Direcții de extindere și îmbunătățire

- Integrarea unei funcționalități pentru salvarea echipelor.
- Suport pentru numere de echipe care nu sunt puteri ale lui 2, prin adăugarea de "byes".
- Dezvoltarea unei versiuni web pentru utilizare extinsă și accesibilitate mai mare.