Gràfica de rendiment

Figure 1: Gràfica de rendiment

Pràctica 1: El problema de l'illa

Implementacions

illa.py (Python)

La implementació en Python funciona correctament en gairebé tots els casos de prova, amb només una excepció coneguda. El programa implementa una solució basada en:

1. Detecció de cicles
2. Cerca en profunditat (DFS)
3. Validació de restriccions d'amistat i gelosia

Cost computacional

- **Cas mitjà**: $O(n^2)$
- **Pitjor cas**: $O(r * n^2)$ on:
  - r: nombre d'arrels possibles
  - n: nombre de nois

illa.hs (Haskell)

La implementació en Haskell és una traducció directa de la versió Python, però actualment presenta alguns errors no identificats. Tot i això, manté una estructura similar a la versió Python:

1. Mateixa lògica de detecció de cicles
2. Mateixa estratègia DFS
3. Gestió similar de les restriccions

Anàlisi de rendiment

La gràfica mostra el temps d'execució de illa.py segons el nombre de nois:

Observacions de la gràfica:

1. **Comportament estable (4-8 nois)**:
   - Temps constant al voltant de 0.36s
   - Indica bona eficiència per casos petits
2. **Salt en rendiment (8-9 nois)**:
   - Augment significatiu a 0.40s
   - Possible punt d'inflexió en la complexitat
3. **Variació moderada (9-12 nois)**:

- Oscil · lacions entre 0.39s i 0.40s
- Manté estabilitat relativa

4. **Creixement exponencial (12-13 nois)**:
   - Augment dramàtic fins a 0.48s
   - Confirma la complexitat $O(n^2)$ teòrica

Limitacions conegudes

illa.py

- Un cas de prova específic no resolt correctament
- Possible millora en l'eficiència per a $n > 12$

illa.hs

- Errors no identificats en l'execució
- Necessita més proves i depuració

Conclusions

1. La implementació en Python és robusta i fiable per a la majoria de casos
2. El rendiment es manté estable fins a 12 nois
3. La versió Haskell necessita més desenvolupament
4. El cost computacional teòric es confirma en la pràctica