Zajęcia D-3: “BFS na nietypowych grafach”

Cel zajęć i efekty uczenia

Główne cele zajęć / materiał do opanowania:

- Szczególny rodzaj grafu - plansza z zablokowanymi polami (*grid graph*)
- Tworzenie grafu w czasie działania programu

Dodatkowe cele:

- [opcjonalnie] Nauka podejścia do długich programów, unikanie powtórzeń kodu

Zadania do rozwiązania na sprawdzarce

Loch czarnoksiężnika

[wersja prostsza zadania - “Loch początkującego czarnoksiężnika”]: Dana jest plansza, na której niektóre pola są wolne, niektóre zablokowane. Jedno pole jest polem startowym i stoi na nim pionek, pewne inne pole jest metą. Pionek może poruszać się tylko po wolnych polach, w każdym ruchu przechodząc o 1 pole. Ile minimalnie potrzeba ruchów, aby pionek dotarł do mety?

[wersja trudniejsza zadania] Pionek dodatkowo jest kolorowy - startuje w kolorze czerwonym. Niektóre pola sprawiają, że pionek zmienia kolor - z czerwonego na zielony i odwrotnie. Z kolei na inne pola można wejść tylko, jeśli pionek ma odpowiedni kolor.

Plan zajęć

Szacunkowy czas trwania: 2 godziny lekcyjne.

1. Wersja prosta zadania: graf planszy, z wierzchołkami jako polami, krawędzie między sąsiednimi polami
2. Implementacja BFS-a na podanym grafie
    - *Należy unikać tworzenia i przechowywania całego grafu - prawdopodobnie i tak nie zmieści się w pamięci operacyjnej. Wierzchołki to pola, a więc pary liczb, które możemy bez zmian trzymać na kolejce. Dla każdego pola pamiętamy, co na nim jest (wolne/zablokowane/meta), oraz czy już było odwiedzone (co jest standardem w algorytmie BFS).*
    - *Po ściągnięciu wierzchołka z kolejki należy przeglądnąć jego wszystkie krawędzie. Nie ma potrzeby ich przechowywać - jeśli wierzchołek to pole (x,y), to jego sąsiadami są pola (x+1, y), (x-1, y), (x, y+1), (x, y-1).*
    - *Dla każdego sąsiada sprawdzamy, czy nie jest zablokowanym polem, i czy nie było odwiedzone. Jeśli nie - dokładamy je do kolejki. To powoduje konieczność powtórzenia cztery razy tego samego ciągu instrukcji - warto zachęcać uczniów, aby nie kopiowali kodu (copy-paste): stwarza to dużą możliwość powielenia*

błędów i jest bardzo niewygodne przy jakiejkolwiek konieczności poprawiania. Jedną z możliwości jest na przykład następująca pętla:

```
ściągnij z kolejki pole (x,y);
vector<int> dx = {1, -1, 0, 0};
vector<int> dy = {0, 0, 1, -1};
for (q = 0; q < 4; q++)
        sprawdź pole (x+dx[q], y+dy[q]);
```

3. Wersja trudniejsza zadania
    - *Traktujemy loch, jakby miał dwa "piętra", tzn. pole jest opisane trzema liczbami (k,x,y) gdzie k to kolor pionka (0 lub 1), a (x,y) położenie. Z pola (k,x,y) można zawsze przejść do sąsiadów (pola (k,x+1,y) etc.). Jeśli pole (k, x, y) jest wirem (zmianą koloru), można wtedy również przejść do pól (1-k, x+1, y), (1-k, x-1, y) itd. Poza tym rozwiązanie jest identyczne.*
    - *W tej wersji kod powtarzałby się 8 razy: tym bardziej należy zachęcać uczniów, aby tego unikali.*
    - *Zadanie w pełnej wersji jest jednym z najtrudniejszych kodersko w całym programie; uczniowie od początku mają tendencję do pisania programów "po swojemu", nieraz w sposób nieuporządkowany, czasem ciężko ich przekonać się do zmiany stylu (robienia wcięć, opisowego nazywania zmiennych, używania funkcji zamiast copy-paste); jedną z funkcji tego zadania jest uświadomienie uczniom, że praktyki dobrego i czytelnego pisania mają sens. Należy się jednak przygotować na to, że wielu uczniów będzie potrzebować pomocy przy poprawianiu swoich programów. Czasem można rozważyć środki bardziej stanowcze: polecić uczniowi, aby napisał swój kod całkiem od nowa, jeśli jest źle zaprojektowany i nieczytelny.*