Zajęcia E-3: “Dwuwymiarowe programowanie dynamiczne”

Cel zajęć i efekty uczenia

Główne cele zajęć / materiał do opanowania:

- Wprowadzenie do wielowymiarowych algorytmów dynamicznych
- Odtwarzanie wyniku przy programowaniu dynamicznym
- Optymalizacja pamięciowa w algorytmach dynamicznych

Dodatkowe cele:

- Różnica między złożonością czasową i pamięciową - podejście teoretyczne i praktyczne

Zadania do rozwiązania na sprawdzarce

Diamenty
Na niektórych polach kwadratowej planszy n x n umieszczone są diamenty. Pionek przechodzi z pola w lewym górnym rogu do pola w prawym dolnym rogu, poruszając się wyłącznie w prawo i w dół. Znaleźć trasę, w której można zebrać w ten sposób najwięcej diamentów.

Neon
Dane sa dwa napisy. Znaleźć najdłuższy podciąg, który występuje jednocześnie w obu (niekoniecznie jako spójny fragment).
Wersja I: podać tylko długość podciągu, mieszcząc się w stosunkowo niskim (np. 8MB) limicie pamięci.
Wersja II: podać cały podciąg.
[opcjonalnie] Wersja III: podać cały podciąg oraz zmieścić się w niskim limicie pamięci.

Plan zajęć

Szacunkowy czas trwania: 4 godziny lekcyjne.

1. Zadanie “Diamenty”
    - *Podzadanie: maksymalna liczba diamentów, którą można zebrać kończąc na polu (i,j).*
    - *Wartość podzadania: W[i][j] = max(W[i-1][j],W[i][j-1]) [+ 1, jeśli na polu (i,j) jest diament] - wynika to z prostej obserwacji, że ścieżka kończąca na polu (i,j) musi wcześniej przejść przez jedno z pól (i-1,j), (i,j-1).*
    - *Odtwarzanie wyniku: na pole (i,j) zawsze wchodzimy z pola (i-1,j) albo (i,j-1) - z tego z nich, które ma większą wartość W. Zaczynając od pola (n,n) i cofając się po jednym ruchu, odtwarzamy całą drogę.*
2. Zadanie “Neon”

- *Problem najdłuższego wspólnego podciągu jest nie tylko ważnym ćwiczeniem z programowania dynamicznego, ale jest fundamentalny ze względu na zastosowania, od podobieństwa kodów DNA po szukanie plagiatów w tekstach.*
- *Podzadanie: dla słów A[1..n] i B[1..m] wartość NWP[i][j] to najdłuższy wspólny podciąg fragmentów A[1..i] oraz B[1..j]. Pełny algorytm jest opisany w bardzo wielu miejscach, nawet na Wikipedii, a także skrótowo zapisany w poniższej ramce:*

```
najdłuższy wspólny podciąg(A[1..n], B[1..m])
        NWP[i][0] = 0, dla każdego i=1,...,n
        NWP[0][j] = 0, dla każdego j=1,...,m
        dla i = 1,..., n:
                dla j = 1,..., m:
                        jeżeli A[i] = B[j]
                                NWP[i][j] = NWP[i-1][j-1]+1
                        w przeciwnym razie:
                                NWP[i][j] = max(NWP[i-1][j],NWP[i][j-1])
```

- *Warto z uczniami przeanalizować całą procedurę na dwóch przykładowych słowach, wypełniając kolejno całą tablicę NWP[i][j] po jednym polu.*

3. Najdłuższy wspólny podciąg - odtwarzanie wyniku
   - *Główna idea - analogicznie jak w "Diamentach", cofamy się od wyniku (NWP[m][n]) po jednym polu.*
     - *Metoda I: w czasie tworzenia tablicy NWP[i][j] zapisujemy w osobnej tablicy R[][], z którego pola ((i-1,j), (i,j-1) lub (i-1,j-1) weszliśmy na (i,j)) - np. R[i][j] = 1, jeśli NWP[i][j] = NWP[i-1][j], 2 jeśli NWP[i][j] = NWP[i][j-1] lub 3, jeśli NWP[i][j] = NWP[i-1][j-1]+1. Mając zapisane te wartości, możemy odtworzyć wynik.*
     - *Metoda II: zauważamy, ze wartości R[i][j] nie musimy mieć zapisanych, możemy je na nowo obliczyć w czasie cofania się, porównując za każdym razem wartość NWP[i][j] z sąsiednimi wartościami.*
4. Najdłuższy wspólny podciąg - optymalizacja pamięciowa
   - *Zauważamy, że przy obliczaniu NWP[i][j] nie korzystamy z wartości NWP[i'][j] dla i'<i-1, możemy więc usunąć je z pamięci. Najprostszą metodą zmodyfikowania oryginalnego algorytmu jest dopisanie "mod 2" do każdego NWP[i][...], jak w poniższej ramce:*

```
najdłuższy wspólny podciąg(A[1..n], B[1..m])
        NWP[i][0] = 0, dla i = 0, 1
        NWP[0][j] = 0, dla każdego j=1,...,m
        dla i = 1,..., n:
                dla j = 1,..., m:
                        jeżeli A[i] = B[j]
```

NWP[i mod 2][j] = NWP[i mod 2][j][j-1]+1
***w przeciwnym razie**:*
NWP[i mod 2][j] =
max(NWP[i mod 2][j],NWP[i mod 2][j-1])

- *Ta technika, niezwykle prosta w implementacji - najpierw napisanie algorytmu z "kwadratową" tablicą, a potem zredukowanie jednego wymiaru za pomocą operacji modulo - działa bardzo często na dwuwymiarowe algorytmy dynamiczne i warto przekazać uczniom, aby ją zapamiętali.*
- *Optymalizacje pamięciowe nie zawsze są potrzebne, ale w wypadku algorytmów dynamicznych często właśnie ten przypadek zachodzi - na przykład dla danych o wielkości ~20 000, złożoność czasowa $O(n^2)$ jest jak najbardziej akceptowalna, ale złożoność pamięciowa $O(n^2)$ nie jest do udźwignięcia dla standardowych komputerów.*
- *Warto zwrócić uwagę na fakt, że w podanej formie optymalizacja pamięciowa i odtwarzanie wyniku wykluczają się ze sobą - odtwarzanie potrzebuje całej tablicy wyników podzadań, nie tylko dwóch ostatnich wierszy. Aby móc pogodzić te techniki ze sobą, konieczne są triki, takie jak algorytm Hirschberga omówiony w następnym punkcie*

5. [opcjonalnie] Algorytm Hirschberga - odtwarzanie wyniku dla najdłuższego wspólnego podciągu, przy jednoczesnej optymalizacji pamięciowej
   - *Algorytm (opisany np. tutaj) nie jest bardzo trudny matematycznie, ale dość złożony i abstrakcyjny - wymaga dobrego opanowania rekursji, a także głębokiego zrozumienia "oryginalnego", opisanego wyżej, algorytmu. Sugeruję stosować go wyłącznie jako dodatkowy materiał dla najambitniejszych uczniów.*