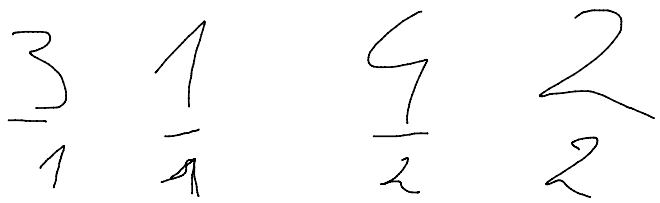
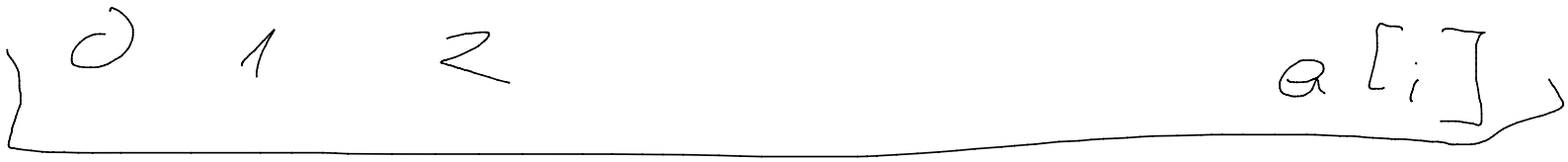
K23 - Dynamiki optymalizacyjne

19 May 2023 17:55

Zacznijmy od jednego z najpopularniejszych problemów optymalizacyjnych. Mamy dany ciąg (a_i) o długości $n \leq 10^6$ i chcemy znaleźć długość jego najdłuższego podciągu rosnącego. Dla proszczenia załóżmy, że $a_i \leqslant 10^6$, gdyż w innym przypadku możemy bezproblemowo przeskalować wartości. Poprzez **podciąg** ciągu (a_i) rozumiemy taki ciąg liczb, który możemy otrzymać z ciągu (a_i) poprzez usunięcie niektórych jego wyrazów.

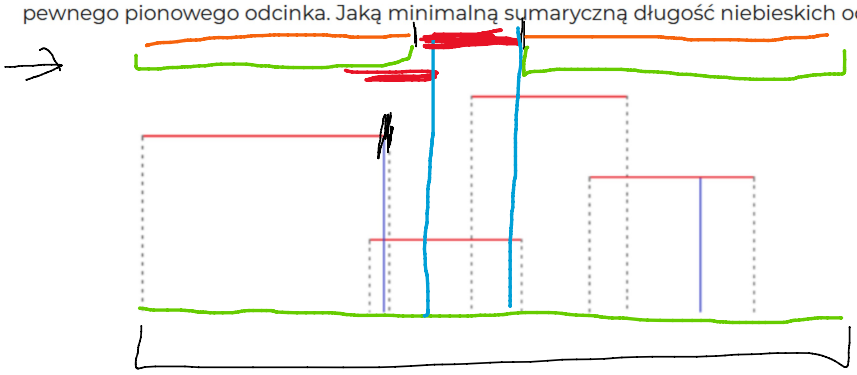


```
int lis(int[] a, int n) {
    int result = 0;
    for (int i = 1; i <= n; i ++) {
        int best = 0;
        for (int j = 1; j < i; j ++)
            if (a[j] < a[i])
                best = max(best, DP[j]);
        DP[i] = best + 1;
        result = max(result, DP[i]);
    }
    return result;
}
```



```
int lis_opt(int[] a, int n) {
    int result = 0;
    PowerTree powerTree = new PowerTree();
    for (int i = 1; i <= n; i ++) {
        int best = powerTree.getMinOnPrefix(a[i] - 1);
        DP[i] = best + 1;
        result = max(result, DP[i]);
        powerTree.insertToTree(a[i], DP[i]);
    }
    return result;
}
```

Pomińmy bajkę o kosmitach i przejdźmy od razu do formalnej treści problemu. Dane jest $n \leq 300$ poziomych (czerwonych) odcinków $(a_i, h_i), (b_i, h_i)$. Chcemy narysować pionowe (niebieskie) odcinki tak, aby każdy z poziomych odcinków dotykał pewnego pionowego odcinka. Jaką minimalną sumaryczną długość niebieskich odcinków możemy uzyskać?



dp[a][b] = sum dł. niebieskich na przedziale a,b

a b

$O(n^2)$ – stanów

$O(n)$ – liczenie

$O(n^3)$

```
int outer_space_invaders(int A[], int B[], int Y[], int n) {
    // Najpierw skalujemy przedziały -
    // powiedzmy, że początek pierwszego jest na pozycji 1,
    // a koniec ostatniego na pozycji d
    for (int len = 1; len <= d; len ++) {
        for (int i = 1; i + len - 1 <= d; i ++) {
            int j = i + len - 1;
            int best = grab(i, j);
            if (best == 0)
                DP[i][j] = 0;
            else {
                DP[i][j] = inf;
                for (int m = A[best]; m <= B[best]; m ++) {
                    int tmp = getDP(i, m - 1);
                    tmp += getDP(m + 1, j);
                    tmp += Y[best];
                    DP[i][j] = min(DP[i][j], tmp);
                }
            }
        }
    }
    return DP[1][DL];
}
```

```
//A[], B[], range[]: charakterystyka czerwonych przedzialow
//DP[][]: tablica z wynikami, na poczatku wypelniona nieskończonościami (inf)
const int inf = 1e9+5;

int grab(intA[], int B[], int Y[], int n, int x, int y)
{
    // Szuka najwyzszego przedzialu, ktory
    // znajduje sie caly w przedziale [x, y]
    int res = 0;
    for (int i = 1; i <= n; i ++)
        if (x <= A[i] && B[i] <= y && Y[res] < Y[i])
    res = i;
    return res;
}

int getDP(int a, int b) {
    // Pomaga nie odwolywac sie do glupich rzeczy
    if (a <= b)
        return DP[a][b];
    return 0;
}
```