Lista 3

Anna Leszczyńska

Styczeń 2026

1 Wstęp

Omówienie algorytmów cut rod, CLS i Huffman.

2 Najciekawsze fragmenty kodu

2.1 CUT ROD

Listing 1: naiwny

```
for (int i = 1; i <= n; ++i)
    { q = max(q, p[i] + naive_cut_rod(p, n - i));
}
```

Ta pętla próbuje każdy możliwy sposób pierwszego cięcia pręta – od ucięcia kawałka długości 1 aż do wzięcia całego pręta bez cięcia. Dla każdego wariantu liczy, ile zarobisz: cena pierwszego kawałka p[i] plus najlepszy możliwy zysk z reszty pręta długości n-i. Na końcu wybiera największy z tych wyników, czyli najlepszy możliwy zysk dla długości n

Listing 2: ze spamietywaniem

```
if (r[n] != -(1LL << 62)) return r[n];
for (int i = 1; i <= n; ++i) {
    ll val = p[i] + memo_rec(p, r, s, n - i);
    if (val > best) {
        best = val;
        best_i = i;
    }
}
```

Jeśli wynik dla długości n jest już zapisany w r[n], funkcja zwraca go od razu. W przeciwnym razie sprawdzane są wszystkie możliwe pierwsze cięcia i=1..n, wybierane jest najlepsze, a wynik jest zapisywany w s[n].

Listing 3: iteracyjny

```

r[0] = 0;
```

```
for (int j = 1; j <= n; ++j) {
    ll best = NEG_INF;
    int best_i = 0;

    for (int i = 1; i <= j; ++i) {
        ll val = p[i] + r[j - i];
        if (val > best) {
            best = val;
            best_i = i;
        }
    }

    r[j] = best;
    s[j] = best_i;
}
```

Dla każdej długości j obliczany jest najlepszy możliwy zysk r[j] poprzez sprawdzenie wszystkich pierwszych cięć i=1..j i użycie wcześniej policzonych wartości r[j-i]. Tablica s[j] zapamiętuje, jakie pierwsze cięcie daje optimum, co pozwala później odtworzyć pełny podział pręta.

2.2 LCS(najdłuższy wspólny podciąg)

Listing 4: LCS iteracyjny

```
        if (X[i - 1] == Y[j - 1]) c[i][j] = c[i - 1][j - 1] + 1;
        else c[i][j] = max(c[i - 1][j], c[i][j - 1]);
```

Są dwie opcje: -litery są takie same (X[i-1] == Y[j-1]) to podciąg wydłuża się, biorąc liczbę "po skosie" (c[i-1][j-1]) i dodając 1 -są różne wtedy liczba jest wybierana poprzez porównanie wartości z góry i z dołu i wybranie większej

Listing 5: LCS rekurencyjny z memoizacją

```
while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs.push_back(X[i - 1]);
            --i; --j;
        } else {
            if (c[i - 1][j] == -1) dp_rec(i - 1, j, X, Y, c);
            if (c[i][j - 1] == -1) dp_rec(i, j - 1, X, Y, c);

            if (c[i - 1][j] >= c[i][j - 1]) --i;
            else --j;
```

Algorytm przechodzi od komórki (i,j) w kierunku (0,0). Jeśli porównywana litera w obu ciągach się jest taka sama , to jest dodana do rozwiązania i następuje przejście po przekątnej. W przeciwnym wypadku porównywane są wartości z komórek sąsiednich (z góry i z lewej), a wybrany zostaje kierunek o większej długości podciągu. Jeżeli dana wartość nie została wcześniej obliczona, jest ona wyznaczana rekurencyjnie z wykorzystaniem memoizacji, co gwarantuje, że każda komórka tablicy dynamicznej jest obliczana co najwyżej jeden raz

2.3 HUFFMAN

Listing 6: Huffman

```
struct Cmp {
    bool operator()(const Node* a, const Node* b) const {
        return a->freq > b->freq;
    }
};
...
priority_queue<Node*, vector<Node*>, Cmp> pq;
```

To gwarantuje, że pq.top() zwraca element o najmniejszej częstotliwości, co minimalizuje długość kodu i powoduje, że algorytm jest zachłanny. Kod jest prefiksowy

Listing 7: ternarnie

```
auto ter = huffman_codes(C, 3);
```

Ta część odpowiada za to że kod jest ternarny (dla binarnego (c, 2)) Drzewo powstające w ternarnym jest płytsze ale szersze niż to w binarnym.