Zadanie 7. (2pkt)

System złożony z dwóch maszyn A i B wykonuje n zadań.
Każde z zadań wykonywane jest na obydwu maszynach, przy czym wykonanie zadania na maszynie B można rozpocząć dopiero po zakończeniu wykonywania go na maszynie A. Dla każdego zadania określone są dwie liczby naturalne a_i, b_i określające czas wykonania i-tego zadania na maszynie A oraz B (odpowiednio). Ułóż algorytm ustawiający zadania w kolejności minimalizującej czas zakończenia wykonania ostatniego zadania przez maszynę B.

Niech x_i oznacza czas „zmarnowany" na oczekiwanie przez maszynę B na wykonanie i-tego zadania przez maszynę A.
Łatwo zauważyć, że zawsze $x_1 = a_1$.
Następnie mamy $x_2 = \max(0, a_1 + a_2 - b_1 - x_1)$, gdyż aby wykonać drugie zadanie na B, najpierw musimy wykonać pierwsze na obu oraz drugie na A oraz odejmujemy x_1, bo nie chcemy 2 razy liczyć tego samego oczekiwania.
Jeśli A wykonało 2 zadania przed pierwszym B, to ta suma jest ujemna, zatem bierzemy czas oczekiwania równy 0 (bo po wykonaniu 1 w B od razu można zrobić 2 w B). Indukcyjnie można wywnioskować stąd wzór:

$$x_n = \max\left(0, \sum_{k=1}^{n} a_k - \sum_{k=1}^{n-1} b_k - \sum_{k=1}^{n-1} x_k\right)$$

Stąd wynika, że:

$$\sum_{k=1}^{n} x_k = \max_{1 \le i \le n}(K_i)$$

Gdzie:

$$K_n = \sum_{k=1}^{n} a_k - \sum_{k=1}^{n-1} b_k$$

Do optymalnego ułożenia stosujemy zasadę Johnsona która mówi, żeby:
0. Inicjalizuj wskaźniki na skrajne pozycje: i = 1, j = n
1. Wybrać najkrótsze zadanie z nieprzydzielonych w A lub B:
a) Jeśli dla niego A < B, to zadanie to wrzucamy na i-tą pozycję oraz i++,
b) Jeśli dla niego A > B, to zadanie to wrzucamy na j-tą pozycję oraz j--,
c) Jeśli dla niego A = B, to nie ma znaczenia czy damy je na początek czy koniec.
2. Usunąć to zadanie z listy zadań zarówno A, jak i B.
3. Powtarzać kroki 1,2 tak długo, aż wszystkie zadania zostaną przydzielone.

Twierdzenia uzasadniające poprawność metody Johnsona:

Fakt 1:
Wykonywanie zadań na obu maszynach w tej samej kolejności jest optymalne.

Jeśli mamy skończone 1 zadanie na maszynie A ale nie na B, to oczywiste jest, że lepiej jest je wykonać również na maszynie B zamiast czekać na inne zadanie z A. Jeśli mamy skończone więcej niż 1 zadanie na maszynie A, ale nie na B, to czas wykonania tych zadań na maszynie B będzie taki sam bez względu na kolejność – np. 3 + 5 = 5 + 3, zatem wtedy taka sama kolejność też jest optymalna.

Fakt 2:
Zadanie j-te jest przed zadaniem j+1 szym, jeśli:

$$(1)\ max(K_j, K_{j+1}) < \max(K_j', K_{j+1}')$$

Gdzie druga z tych sekwencji powstała poprzez zamienienie kolejności wykonania zadań j-tego z j+1 szym.

Dla przypomnienia: $K_j = \sum_{k=1}^{j} a_k - \sum_{k=1}^{j-1} b_k$, zatem odejmując obustronnie od (1) wyraz $\sum_{k=1}^{j+1} a_k - \sum_{k=1}^{j-1} b_k$ otrzymujemy:

$$max(-a_{j+1}, -b_j) < \max(-a_j, -b_{j+1})$$

Mnożąc to przez -1 otrzymujemy:

$$(2)\ \min(a_j, b_{j+1}) < min(a_{j+1}, b_j)$$

Fakt 3:
Relacja (2) z faktu 2 jest przechodnia.
Załóżmy, że $\min(a_1, b_2) \leq min(a_2, b_1)$ oraz $\min(a_2, b_3) \leq min(a_3, b_2)$.
Wtedy $\min(a_1, b_3) \leq min(a_3, b_1)$ za wyjątkiem gdy wszystkie 3 są równe.

Rozważmy 4 przypadki:
1. $a_1 \leq a_2, b_1, b_2$ oraz $a_2 \leq a_3, b_2, b_3$
Wtedy $a_1 \leq a_2 \leq a_3$ oraz $a_1 \leq b_1$, stąd otrzymujemy

$$L = \min(a_1, b_3) = a_1 \leq \min(a_3, b_1) = P \blacksquare$$

2. $b_2 \leq a_1, a_2, b_1$ oraz $b_3 \leq a_2, a_3, b_2$
Wtedy $b_3 \leq b_2 \leq b_1$ oraz $b_3 \leq a_3$, stąd otrzymujemy

$$L = \min(a_1, b_3) = b_3 \leq \min(a_3, b_1) = P \blacksquare$$

3. $a_1 \leq a_2, b_1, b_2$ oraz $b_3 \leq a_2, a_3, b_2$
Wtedy $a_1 \leq b_1$ oraz $b_3 \leq a_3$, stąd otrzymujemy
$$L = \min(a_1, b_3) \leq \min(a_3, b_1) = P \blacksquare$$
4. $b_2 \leq a_1, a_2, b_1$ oraz $a_2 \leq a_3, b_2, b_3$
Wtedy $b_2 \leq a_2$ oraz $a_2 \leq b_2$, czyli $a_2 = b_2$, stąd z założenia otrzymujemy $\min(a_1, b_2) = \min(a_1, a_2) = min(a_2, a_1) = min(a_2, b_1)$, podobnie dla a3, zatem mamy 3 identyczne pary, co jest sprzecznością z założeniem.

Pseudokod:

```
struct Job
{
    int a, b;
    bool operator<(Job o) const
    {
        return min(a, b) < min(o.a, o.b);
    }
};

// ustala kolejność zadań maszyn
vector<Job> johnsons_rule(vector<Job> jobs)
{
    sort(jobs.begin(), jobs.end());
    vector<Job> a, b;
    for (Job j : jobs)
    {
        if (j.a < j.b)
            a.push_back(j);
        else
            b.push_back(j);
    }
    a.insert(a.end(), b.rbegin(), b.rend());
    return a;
}
```