1 Treść zadania

(Konserwatny MM) Zaproponuj modyfikację algorytmu Maximal Matching przedstawionego w notatkach do wykładu, która umożliwi jego działanie przy dodatkowym założeniu, że każdy proces jest typu A albo B i dwa dwa procesy tego samego typu nie mogą tworzyć pary. Ponadto możesz przyjąć, że typ danego procesu jest z góry zadany i nigdy nie może zostać zmieniony. Uzasadnij poprawność i zbieżność algorytmu. UWAGA: Jeśli nie zaznaczono inaczej, wszystkie definicje i oznaczenia są tożsame z wykładem

2 Rozwiązanie

W celu przedstawienia typu procesu wprowadzimy jednoargumentowy predykat *typ*.

$$typ(p) = \{A, B\} \tag{1}$$

Dla każdego procesu p. Na podstawie powyższego możemy zmodyfikować warunki dodając do nich sprawdzenie $typ(p) \neq typ(q)$, jak pokazano poniżej

$$if\, pref_p = NULL \ \wedge \ (\exists q \in N(p))(pref_p = p) \ \wedge \ typ(p) \neq typ(q) \tag{2}$$

Analogicznej modyfikacji poddajemy pozostałe warunki algorytmu oraz wszystkie możliwe stany procesu.
Wskazane modyfikacje zapobiegają sytuacjom, w których dwa procesy tego samego typu zostałyby ze sobą połączone w ramach mechanizmu tworzenia maksymalnego dopasowania.
TL;DR Nastepnych dwóch sekcji: Dowody są dość analogiczne do tych, wykonanych w poprzednich zadaniach, bo wprowadzone zmiany niezbyt wpływają na cały mechanizm tworzenia maksymalnego dopasowania- dodane warunki możemy przenieść do definicji sąsiedztwa (znamy z góry typ każdego z procesów, typ procesu nie może się zmieniać w czasie)

$$N(p) = \{q : \{p, q\} \in E \ \wedge \ typ(p) \neq typ(q)\} \tag{3}$$

i dowody są identyczne

3 Uzsadnienie poprawności

Zgodnie z Lemat 3 z wykładu: Zachodzą warunki specyfikacji S $\iff$ konfiguracja jest ostateczna (tzn. żaden krok algorytmu nie może zmienić konfiguracji)
Dowód: ($\Rightarrow$) Pokażemy, że jeśli zachodzą warunku S to żadna akcja algorytmu nie jest możliwa.
Przypomnienie: Specyfikację S definiujemy następująco: w każdej poprawnej konfiguracji prawdziwe jest zdanie

$$(\forall p)(married(p) \ \vee \ single(p)) \tag{4}$$

Rozpatrzmy po kolei wszystkie możliwe kroki jakie może wykonać algorytm dla danego procesu:

- (Accept proposal) Aby warunek został spełniony, proces p musi nie mieć dopasowanej żadnej pary (być w stanie *free*) oraz musi istnieć stan q o innym typie, który może przyjąć p jako parę (czyli musi być w stanie *wait*). Zgodnie z założeniem, aktualnie posiadamy procesy tylko w dwóch stanach: *single* lub *married*, więc warunek nie zostanie nigdy spełniony
- (Propose) Aby warunek został spełniony, proces p musi nie mieć dopasowanej żadnej pary (być w stanie *free*), żaden z procesów q o innym typie nie może zadeklarować p jako swoją parę (wtedy zaakceptowalibysmy propozycję) oraz musi istnieć proces q o innym typie bez preferencji, czyli q również musiałby być w stanie *free*. Procesy mogą być tylko w stanie *single* lub *married*, wykluczamy
- (Unchain) - proces p musi byc w stanie *chain*, wykluczamy

Wniosek - gdy konfiguracja S spełniona, wykonanie ruchu jest niemożliwe
($\Leftarrow$) Dokonamy sprawdzenia co się stanie z procesem, który nie jest w stanie *married* bądź *single*. Bez straty ogólności wybieramy proces p zgodnie z powyższym i dokonujemy sprawdzeń:

- Jeśli proces p jest w stanie *free* to może:
  - Przyjąć propozycję od procesu q o innym typie w stanie *wait*
  - Dokonać *unchain* jeśli q o innym typie jest w stanie *chain*
  - Złożyć propozycję, jeśli q jest w stanie *free*
- Jeśli proces p jest w stanie *chain* to dokona *unchain*
- Jeśli proces p jest w stanie *wait* to musi istniec proces o stanie *free* i przyjmuje propozycję

Jeśli istnieją stany poza *single* oraz *married* w konfiguracji to zawsze można wykonać ruch, czyli nie jest ona ostateczna.

4 Dowód zbieżności

Rozważmy funkcję potencjału $F(t) = (c, f, w, m + s)$ gdzie:

- c - liczba procesów w stanie *chain*
- f - liczba procesów w stanie *free*
- w - liczba procesów w stanie *wait*
- m+s - liczba procesów w stanie *married* lub *single*

Sprawdźmy zmianę każdego z argumentów przy każdej możliwej egzekucji algorytmu dla jednego procesu:

- Accept proposal: $free(p)$ i $wait(q)$ zmienia się na $married(p)$ i $married(q)$ (lub single, jeśli nie może w married, ale nas to zbytnio nie interesuje, bo grupujemy married i single razem)

$$(c, f-1, w-1, m+s+2) \prec (c, f, w, m+s) \tag{5}$$

  Element na drugim miejscu zmniejsza się przy zachowaniu wartości pierwszego. Stąd wnioskujemy, iż:

$$F(t+1) \prec F(t) \tag{6}$$

- Proposal: $free(p)$ i $free(q)$ zamieniają się w $wait(p)$ i $free(q)$

$$(c, f-1, w+1, m+s) \prec (c, f, w, m+s) \tag{7}$$

  Element na drugim miejscu zmniejsza się przy zachowaniu wartości pierwszego. Stąd wnioskujemy, iż:

$$F(t+1) \prec F(t) \tag{8}$$

- Unchain: $chain(p)$ zmienia się na $free(p)$

$$(c-1, f+1, w, m+s) \prec (c, f, w, m+s) \tag{9}$$

  gdyż element na pierwszym miejscu zmniejsza się. Stąd wnioskujemy, iż:

$$F(t+1) \prec F(t) \tag{10}$$

Zauważamy, że porządek leksykograficzny jest dobrym porządkiem, czyli ma wartość najmniejszą, która w naszym przypadku wynosi $(0, 0, 0, n)$, czyli wszystkie procesy są *married* lub *single*. Jak sprawdzić złożoność obliczeniową? Badając najgorszy przypadek: zaczynamy ze wszystkimi procesami w stanie chain.

$$F(t) = (n, 0, 0, 0) \tag{11}$$

W najgorszym przypadku Deamon odpala każdy z procesów po kolei. Wtedy wszystkie dokonują *unchain* w n krokach

$$F(t) = (0, n, 0, 0) \tag{12}$$

Powtarzamy powyższe rozumowanie dla propozycji

$$F(t) = (0, 0, n, 0) \tag{13}$$

Oraz dla akceptacji propozycji

$$F(t) = (0, 0, 0, n) \tag{14}$$

Wykonwaliśmy $n+n+n = 3n$ kroków, aby z najgorszego ułożenia dostać się do takiego, które spełnia zadaną konfigurację. Wiemy, iż $O(n^3)$ jest o wiele większe niż 3n, więc pokazaliśmy, że algorytm osiąga konfigurację legalną w nie więcej niż $O(n^3)$ krokach.