Projekt – Symulacja sieci

W ramach tego zadania zaprojektujesz system służący do modelowania i symulacji działania sieci – na przykładzie “linii produkcyjnych” w fabryce. Następnie wykorzystasz poznaną wiedzę z zakresu technik programowania obiektowego w C++, aby zaimplementować zaprojektowany system.

Model

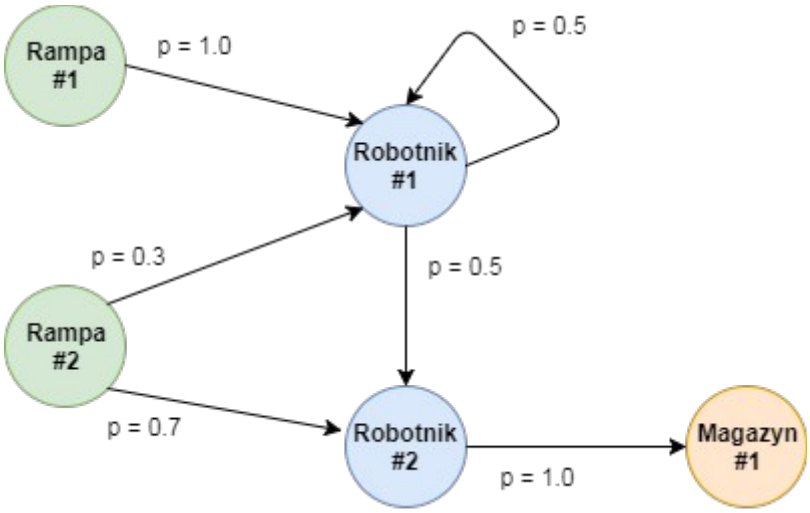
Linia produkcyjna w fabryce (**sieć**, ang. net) składa się z następujących elementów (**węzłów**, ang. nodes), połączonych w spójną całość:

- rampy rozładunkowe (ang. loading ramps) – dostarczanie półproduktów (**źródła**, ang. sources)
- robotnicy (ang. workers) – przetwarzanie półproduktów
- magazyny (ang. storehouses) – przechowywanie wytworzonych półproduktów (**ujścia**, ang. sinks)

Dopuszczalne są następujące **połączenia** (ang. links) w ramach procesu technologicznego:

- rampa rozładunkowa → robotnik (dostarczenie półproduktów do pierwszego etapu produkcji)
- robotnik → robotnik (dostarczenie półproduktu do dalszej obróbki; w szczególności dopuszczamy sytuację, gdy ten sam robotnik przekazuje efekt swojej pracy samemu sobie – do dalszej obróbki)
- robotnik → magazyn (umieszczenie półproduktu w magazynie)

Każda rampa i każdy robotnik może przekazywać półprodukty tylko wybranego (własnego) podzbioru wszystkich możliwych odbiorców (robotników i magazynów).



Przykładowa sieć

Dodatkowo przyjmujemy następujące założenia:

- Dostawy półproduktów do ramp odbywają się z określoną częstotliwością, przy czym jednorazowo dostarczana jest jedna jednostka półproduktu.
- Jednorazowo robotnik przetwarza pojedynczą jednostkę półproduktu, przy czym każdemu robotnikowi proces ten zajmuje określoną ilość czasu (może się ona różnić pomiędzy robotnikami, lecz zawsze wynosi co najmniej jedną jednostkę czasu).
- Każdy robotnik posiada swoje preferencje odnośnie kolejności przetwarzania dostarczanych mu półproduktów – np. jedni wolą przetwarzać je w kolejności napływania (kolejka *first in, first out*, FIFO), inni wolą zaczynać od "najświeższej" dostarczonego półproduktu (kolejka *last in, first out*, LIFO).
- Domyślnie każdy robotnik przekazuje przetworzony półprodukt jednemu z przypisanych mu odbiorców nie faworyzując żadnego z nich (tj. prawdopodobieństwo, że robotnik przekaże przetworzony półprodukt do danego odbiorcy, jest jednakowe dla każdego odbiorcy). Czasem jednak robotnik może mieć w tym względzie swoje własne preferencje (np. będzie przekazywał swe wyroby robotnikowi A dwa razy częściej niż robotnikowi B). Przyjmij, że na rampie rozładunkowej również pracuje osoba, która może mieć swoje preferencje… Preferencje danego robotnika w kwestii wyboru odbiorców danego półproduktu można traktować jako dyskretny rozkład prawdopodobieństwa – suma prawdopodobieństw poszczególnych zdarzeń $p(Xi)$

("dany przetworzony półprodukt zostanie przekazany do odbiorcy X o ID = i") musi wynosi 1 (czyli np. dla puli trzech odbiorców – robotnicy #1 i #2 oraz magazyn #1 – dopuszczalnym rozkładem będzie: $p(R1)=0.2$, $p(R2)=0.3$, $p(S1)=0.5$

-).
- Dodanie nowego odbiorcy do puli odbiorców danego węzła bez precyzowania prawdopodobieństwa powoduje ustawienie preferencji na rozkład jednostajny. Dodanie odbiorcy wraz ze zdefiniowaniem prawdopodobieństwa powoduje takie przeskalowanie dotychczasowych preferencji, aby nowy zbiór preferencji stanowił rozkład prawdopodobieństwa.
  Przykład (dodawanie odbiorcy ze zdefiniowanym prawdopodobieństwem):

- dotychczasowe preferencje: $p(R1)=0.4$, $p(R2)=0.6$ ($\sum p(X)=1$
-)
- dodawany odbiorca: $p(S1)=0.2$
- nowe preferencje: $p(R1)=(1-0.2)\times 0.4=0.32$, $p(R2)=(1-0.2)\times 0.6=0.48$, $p(S1)=0.2$ ($\sum p(X)=1$
-)

- Usunięcie odbiorcy powoduje takie przeskalowanie pozostałych preferencji, aby nowy zbiór preferencji stanowił rozkład prawdopodobieństwa.
  Przykład:

- dotychczasowe preferencje: $p(R1)=0.32$, $p(R2)=0.48$, $p(S1)=0.2$ ($\sum p(X)=1$

-)
- usuwany odbiorca: $S1$
- nowe preferencje: $p(R1)=0.32/(1-0.2)=0.4$, $p(R2)=0.48/(1-0.2)=0.6$ ($\sum p(X)=1$
- Każdy półprodukt dostarczany do fabryki ma swoją unikalną etykietę, która nie zmienia się pomiędzy etapami produkcji.

Symulacja

Dla zadanego modelu sieci (wczytanego z pliku) chcemy mieć możliwość zasymulowania jej działania – np. aby sprawdzić, jaka jest potencjalna wydajność linii produkcyjnej.

Symulacja składa się z **tur** (ang. rounds), odpowiadających zdarzeniom w systemie, które zaszły w danej jednostce czasu (czas w symulacji jest skwantowany – przyjmij, jedna jednostka czasu odpowiada 1 godzinie). Każda tura składa się z następujących **etapów** (ang. stages), rozpatrywanych w kolejności:

1. Dostawy półproduktów do ramp.
2. Przekazywanie półproduktów do odbiorców.
3. Przetwarzanie półproduktów przez robotników.

(A więc w ramach pojedynczej tury ten sam półprodukt może być kolejno rozładowany na rampie i przekazany do robotnika do obróbki.)

Przyjmij, że w ramach każdego etapu symulacji robotnicy wykonują pracę sekwencyjnie, w dowolnej kolejności (czyli np. najpierw pracę wykonuje robotnik #1, potem robotnik #2 itd.), a nie jednocześnie.

Przed rozpoczęciem właściwej symulacji należy zweryfikować to, czy utworzona sieć jest spójna – tzn. czy każda jednostka półproduktu, rozładowana na obojętnie z której ramp, zostanie ostatecznie (po ewentualnym przetworzeniu) umieszczona w magazynie. W przypadku błędnej sieci funkcja weryfikująca powinna rzucić wyjątek zawierający opis przyczyny błędu (np. *“Robotnik #2 nie posiada żadnego połączenia wyjściowego.”*).

Raportowanie

Tworzony system powinien umożliwiać dwie opcje raportowania:

- raport o strukturze sieci
- raport o stanie symulacji

Wygenerowane raporty mogą być zapisywane do pliku tekstowego bądź na standardowe wyjście (w naszym przypadku – do konsoli).

Powinna istnieć możliwość generowania raportu o stanie symulacji w następujących trybach:

- dla zadanych tur (np. dla trzeciej i piątej tury, dla ostatniej tury)
- w zadanych odstępach czasu (np. co drugą turę)

Zakres funkcjonalności aplikacji

Zaimplementowana przez Ciebie aplikacja do symulowania sieci powinna wykonywać kolejno następujące operacje:

1. **Wczytanie struktury sieci z pliku.**
   Ewentualne błędy parsowania pliku powinny być sygnalizowane z użyciem mechanizmu wyjątków i obsłużone w głównej funkcji.
2. **Dokonanie ewentualnych modyfikacji struktury sieci i zapis nowej struktury do pliku.**
3. **Weryfikacja poprawności wczytanej sieci.**
   Ewentualne błędy parsowania pliku powinny być sygnalizowane z użyciem mechanizmu wyjątków i obsłużone w głównej funkcji.
4. **Symulacja działania sieci wraz z raportowaniem.**

Specyfikacje plików

Poniżej znajdują się specyfikacje formatu danych użytych w plikach konfiguracyjnych oraz raportach.

 Zapoznanie się ze specyfikacjami będzie niezbędne dopiero w fazie implementacji – *nie* w fazach koncepcyjnej i projektowania interfejsów!

Specyfikacja pliku wejściowego zawierającego strukturę sieci

Plik wejściowy zawiera opis struktury sieci w postaci tekstowej. Każda linia pliku opisuje jeden element sieci – węzeł (rampę załadunkową, robotnika, bądź magazyn) albo połączenie między węzłami:

- **rampa załadunkowa**

  ```
  LOADING_RAMP id=<ramp-id> delivery-interval=<delivery-interval>
  ```

  - `ramp-id`: unikalny identyfikator rampy – w ramach ramp (liczba całkowita)
  - `delivery-interval`: częstotliwość dostaw jako okres pomiędzy dostawami wyrażony w jednostkach czasu (liczba całkowita)

- **robotnik**

  ```
  WORKER id=<worker-id> processing-time=<processing-time> queue-type=<queue-type>
  ```

  - `worker-id`: unikalny identyfikator robotnika – w ramach robotników (liczba całkowita)
  - `processing-time`: czas przetwarzania półproduktu wyrażony w jednostkach czasu (liczba całkowita)
  - `queue-type`: preferencje przetwarzania dostarczonych półproduktów (typ wyliczeniowy: `LIFO`, `FIFO`)

- **magazyn**

  ```
  STOREHOUSE id=<storehouse-id>
  ```

  - `storehouse-id`: unikalny identyfikator magazynu – w ramach magazynów (liczba całkowita)

- **połączenie**

  ```
  LINK src=<node-type>-<node-id> dest=<node-type>-<node-id> p=<probability>
  ```

  - `node-type`: kategoria węzła (typ wyliczeniowy: `ramp`, `worker`, `store`)
  - `node-id`: identyfikator węzła w ramach danej kategorii (liczba całkowita)
  - `probability`: preferencje wyboru danego odbiorcy – prawdopodobieństwo zdarzenia (liczba zmiennoprzecinkowa z zakresu ⟨0,1⟩
  - , zapis z kropką dziesiętną)

Elementy w pliku występującą w następującej kolejności: `LOADING_RAMP`, `WORKER`, `STOREHOUSE`, `LINK`.

Kolejność obiektów w ramach danego typu elementów jest dowolna (nie muszą być posortowane zgodnie ze wzrastającym identyfikatorem).

Puste linie oraz linie zaczynające się od średnika (`;`) są pomijane.

Specyfikacja struktury sieci z przykładu

struct-input.txt

```
; == LOADING RAMPS ==

LOADING_RAMP id=1 delivery-interval=3
LOADING_RAMP id=2 delivery-interval=2

; == WORKERS ==

WORKER id=1 processing-time=2 queue-
type=FIFO
WORKER id=2 processing-time=1 queue-
type=LIFO

; == STOREHOUSES ==

STOREHOUSE id=1

; == LINKS ==

LINK src=ramp-1 dest=worker-1 p=1.0

LINK src=ramp-2 dest=worker-1 p=0.3
LINK src=ramp-2 dest=worker-2 p=0.7

LINK src=worker-1 dest=worker-1 p=0.5
LINK src=worker-1 dest=worker-2 p=0.5

LINK src=worker-2 dest=store-1 p=1.0
```

Raport o strukturze sieci

Plik raportu o strukturze sieci zawiera jej opis w postaci tekstowej. Plik składa się z sekcji grupujących poszczególne kategorie węzłów, a każda z sekcji zawiera bloki opisujące poszczególne węzły w ramach danej kategorii węzłów.

Sekcje sformatowane są w następującej postaci (zwróć uwagę na puste linie):

```
== <section-name> ==

<node_1>

<node_2>

...

<node_N>
```

- `section-name`: nazwa sekcji (typ wyliczeniowy: `LOADING RAMPS`, `WORKERS`, `STOREHOUSES`)
- `node_i`: blok opisujący i-ty węzeł (w ramach danej kategorii węzłów)

Sekcje występują w pliku wyjściowym w następującej kolejności: `LOADING RAMPS`, `WORKERS`, `STOREHOUSES`.

Format bloków:

- **rampa załadunkowa**

  ```
  LOADING RAMP #<ramp-id>
    Delivery interval: <delivery-interval>
    Receivers:
      <receiver_1>
      <receiver_2>
      ...
      <receiver_N>
  ```

  - `ramp-id`: unikalny identyfikator rampy – w ramach ramp (liczba całkowita)
  - `delivery-interval`: częstotliwość dostaw jako okres pomiędzy dostawami wyrażony w jednostkach czasu (liczba całkowita)
  - `receiver_i`: wpis określający i-tego odbiorcę (zob. niżej)

- **robotnik**

  ```
  WORKER #<worker-id>
    Processing time: <processing-time>
    Queue type: <queue-type>
    Receivers:
      <receiver_1>
      <receiver_2>
      ...
      <receiver_N>
  ```

  - `worker-id`: unikalny identyfikator robotnika – w ramach robotników (liczba całkowita)

    - `processing-time`: czas przetwarzania półproduktu wyrażony w jednostkach czasu (liczba całkowita)
    - `queue-type`: preferencje przetwarzania dostarczonych półproduktów (typ wyliczeniowy: `LIFO`, `FIFO`)
    - `receiver_i`: wpis określający i-tego odbiorcę (zob. niżej)
- **magazyn**

  ```
  STOREHOUSE #<storehouse-id>
  ```

    - `storehouse-id`: unikalny identyfikator magazynu – w ramach magazynów (liczba całkowita)
- **odbiorca**

  ```
  <node-type> #<node-id> (p = <probability>)
  ```

    - `node-type`: kategoria węzła (typ wyliczeniowy: `ramp`, `worker`, `storehouse`)
    - `node-id`: identyfikator węzła w ramach danej kategorii (liczba całkowita)
    - `probability`: preferencje wyboru danego odbiorcy – prawdopodobieństwo zdarzenia (liczba zmiennoprzecinkowa z zakresu ⟨0,1⟩
    - , zapis z kropką dziesiętną)

Każdy poziom wcięcia jest wyrażany poprzez dwie spacje (a *nie* tabulatory).

Bloki opisujące węzły w danej kategorii powinny być posortowane zgodnie ze wzrastającym identyfikatorem.

Raport o strukturze dla sieci z przykładu
struct-report.txt

```
== LOADING RAMPS ==

LOADING RAMP #1
  Delivery interval: 3
  Receivers:
    worker #1 (p = 1.0)

LOADING RAMP #2
  Delivery interval: 2
  Receivers:
    worker #1 (p = 0.3)
    worker #2 (p = 0.7)


== WORKERS ==

WORKER #1
  Processing time: 2
  Queue type: FIFO
  Receivers:
    worker #1 (p = 0.5)
    worker #2 (p = 0.5)
```

Raport o strukturze dla sieci z przykładu

```
WORKER #2
  Processing time: 1
  Queue type: LIFO
  Receivers:
    storehouse #1 (p = 1.0)


== STOREHOUSES ==

STOREHOUSE #1
```

Raport o stanie symulacji

Plik raportu o stanie symulacji zawiera jego opis w postaci tekstowej. Plik składa się z sekcji grupujących poszczególne kategorie węzłów, a każda z sekcji zawiera bloki opisujące poszczególne węzły w ramach danej kategorii węzłów.

Sekcje sformatowane są w następującej postaci (zwróć uwagę na puste linie):

```
== <section-name> ==

<node_1>

<node_2>

...

<node_N>
```

- `section-name`: nazwa sekcji (typ wyliczeniowy: `WORKERS`, `STOREHOUSES`) (podane w kolejności występowania w pliku wyjściowym).
- `node_i`: blok opisujący i-ty węzeł w ramach danej kategorii węzłów (zob. niżej)

Format bloków:

- **robotnik**

  ```
  WORKER #<worker-id>
    Queue: #<element_1-id> (pt = <processing-time>), #<element_2-id>, ...,
  #<element_N-id>
  ```

  - `worker-id`: unikalny identyfikator robotnika – w ramach robotników (liczba całkowita)
  - `element_i-id`: etykieta i-tego półproduktu znajdującego się w kolejce (liczba całkowita) [zob. Model – dodatkowe założenia]
  - `processing-time`: czas spędzony dotychczas przez robotnika na przetwarzaniu (aktualnie przetwarzanego) półproduktu

- **magazyn**

  ```
  STOREHOUSE #<storehouse-id>
    Queue: #<element_1-id>, #<element_2-id>, ..., #<element_N-id>
  ```

- `storehouse-id`: unikalny identyfikator magazynu – w ramach magazynów (liczba całkowita)
- `element_i-id`: etykieta i-tego półproduktu znajdującego się w kolejce (liczba całkowita) [zob. Model – dodatkowe założenia]

Każdy poziom wcięcia jest wyrażany poprzez dwie spacje (a *nie* tabulatory).

Bloki opisujące węzły w danej kategorii powinny być posortowane zgodnie ze wzrastającym identyfikatorem.

Przykładowy raport stanu symulacji dla sieci z przykładu
sim-report.txt

```
== WORKERS ==

WORKER #1
  Queue: #1 (pt = 1), #5

WORKER #2
  Queue: #3 (pt = 1)


== STOREHOUSES ==

STOREHOUSE #1
  Queue: #2, #4
```