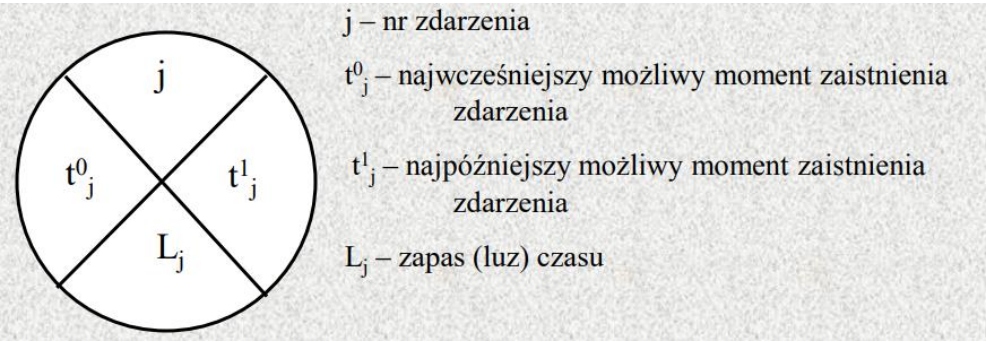
Wydział: WIMiIP	**Imiona i nazwiska:** Tomasz Artecki Kacper Bielak Dominik Budzowski	**Kierunek studiów:** Informatyka Techniczna	**Rok:** 3 (semestr VI)	**Grupa projektowa:** 1
Temat: Metoda CPM				
Data oddania: 12.04.2022	**Przedmiot:** Badania operacyjne i logistyka			**Nr zespołu:** 2

Wstęp teoretyczny metody CPM

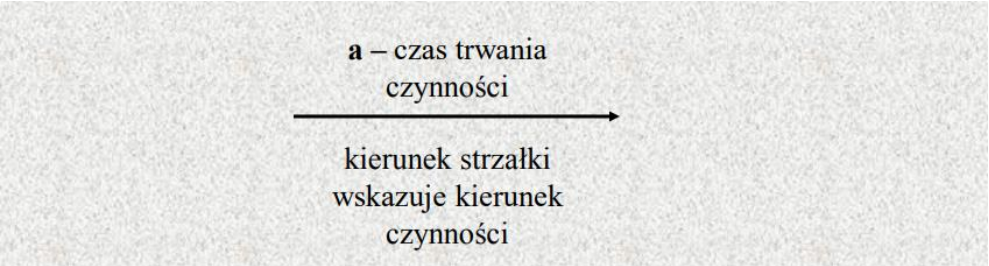
Metoda ścieżki krytycznej (ang. Critical Path Method, CPM) to technika, w której określa się zadania wymagane do ukończenia projektu oraz rezerwy czasowe harmonogramu. Ścieżka krytyczna w zarządzaniu projektami jest najdłuższą sekwencją czynności, które należy wykonać w terminie, dla ukończenia projektu. Wszelkie opóźnienia zadań krytycznych doprowadzą do opóźnienia całego projektu. Polega ona na określeniu najważniejszych zadań na osi czasu projektu, wyznaczeniu zależności zadań oraz obliczeniu czasu trwania zadań. Metoda CPM została stworzona pod koniec lat 50. jako rozwiązanie problemu zwiększonych kosztów wynikających z tworzenia niewydajnych harmonogramów. Od tego czasu metoda CPM stała się popularnym narzędziem do planowania projektów i określania priorytetu zadań. Ułatwia ona rozłożenie skomplikowanych projektów na poszczególne zadania i określenie elastyczności projektu.

W metodzie CPM rozróżniamy dwa komponenty:

- ✓ Zdarzenie – moment rozpoczęcia lub zakończenia jednej lub większej liczby czynności (przedstawiane graficznie jako kółko):



- ✓ Czynność - dowolnie wyodrębniony element przedsięwzięcia opisany czasem trwania w ramach którego zużywane są określone środki (przedstawiana graficznie jako strzałka):



Podczas tworzenia sieci obowiązują następujące reguły:

- ✓ Zdarzenie początkowe nie ma czynności poprzedzających;
- ✓ Zdarzenie końcowe nie ma czynności następujących;
- ✓ Dwa kolejne zdarzenia mogą być połączone tylko jedną czynnością;
- ✓ Jeżeli czynność A jest bezpośrednim poprzednikiem czynności B to węzeł końcowy (zdarzenie) czynności A staje się węzłem początkowym czynności B;
- ✓ Jeżeli czynność X ma kilku poprzedników to końcowe węzły (zdarzenia) tych czynności są reprezentowane tylko przez jeden węzeł, który jest węzłem początkowym czynności X;
- ✓ Jeżeli czynność X jest poprzednikiem dla kilku czynności to końcowy węzeł czynności X jest węzłem początkowym dla tych czynności.

Cechy charakterystyczne sieci:

1. Ścieżka krytyczna – określa najdłuższy czas przejścia sieci (czynności na niej mają zerowy zapas czasu).

2. Najwcześniejszy możliwy moment zaistnienia j-tego zdarzenia „t_j^0 ”.

   $t_j^0 = t_i^0 + t_{i-j}$ t_{i-j} – czas trwania czynności między zdarzeniami ‘i’ oraz ‘j’

   Jeżeli do zdarzenia dochodzi więcej niż jedna czynność to najwcześniejszy możliwy moment zaistnienia tego zdarzenia jest równy maksymalnej z obliczonych wielkości:

   $$t_j^0 = \max\{t_i^0 + t_{i-j}\}, \text{ i<j}$$

   i – zdarzenia poprzedzające j - zdarzenia następujące

3. Najpóźniejszy możliwy moment zaistnienia j-tego zdarzenia t_j^1.

   Najpóźniejszy moment zaistnienia zdarzenia końcowego jest równy najwcześniejszemu momentowi zaistnienia tego zdarzenia. Jego wyznaczanie realizowane jest od końcowego do początkowego zdarzenia w sieci. Jeżeli do zdarzenia dochodzi więcej niż jedna czynność to najpóźniejszy możliwy moment zaistnienia tego zdarzenia jest równy minimalnej z obliczonych wielkości:

   $$t_j^0 = \min\{t_i^1 - t_{i-j}\}, \text{ j<i}$$

   j – zdarzenia poprzedzające i - zdarzenia następujące

4. Zapas (luz) czasowy.

Określa jak bardzo można opóźnić moment zaistnienia danego zdarzenia bez wpływu na termin zakończenia realizacji projektu.

$$L_j = t_j^1 - t_j^0$$

Temat projektu

Tematem projektu jest zbudowanie aplikacji webowej umożliwiającej wykonywanie głównych operacji związanych z metodą CPM. Aplikacja webowa w oparciu o podane czynności rysuje graf CPM i harmonogram Gantta.

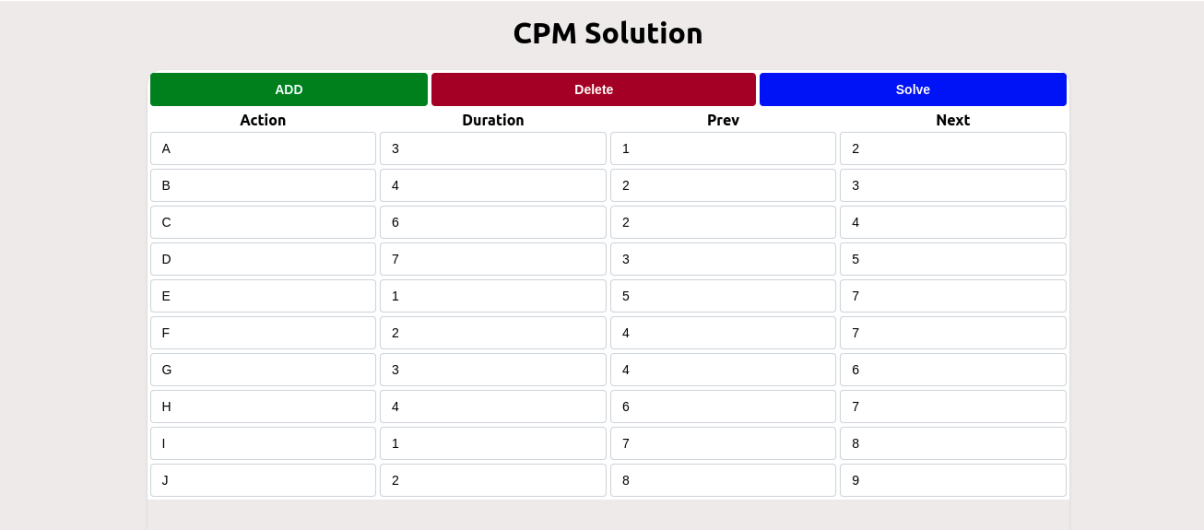
Użyte narzędzia informatyczne

Aplikacja została napisana przy użyciu języka Typescript w środowisku node.

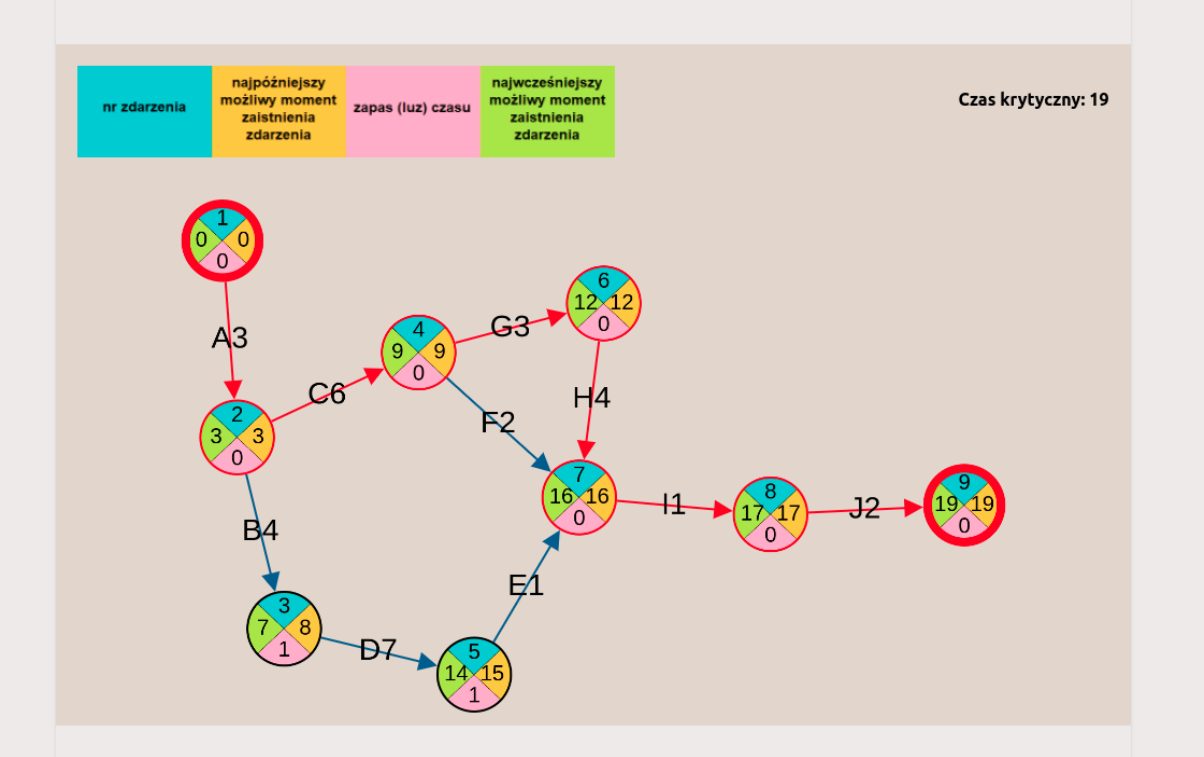
Użyte biblioteki:	
Serwer:	- **express** (główne środowisko back-endowe) - **cors** (do łączenia serwera z klientem)
Klient:	- **react** (główne środowisko front-endowe) - **axios** (do komunikacji z backendem) - **react-router-dom** (do ustawienia adresacji) - **react-dom** (rozszerzenie react'a) - **cytoscape** (do narysowania grafu CPM) - **cytoscape-cose-bilkent** (do layoutu grafu CPM) - **react-google-charts** (do rysowania harmonogramu Gantta)

Prezentacja projektu

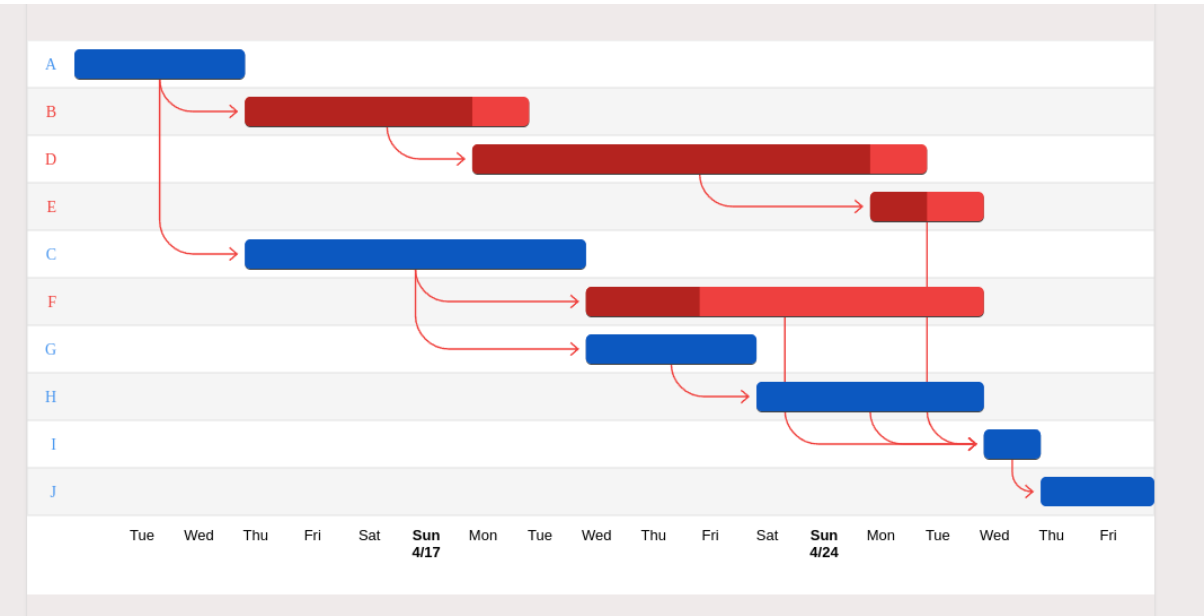
Zrzut ekranu przedstawiający interfejs do wprowadzania danych wejściowych:



Zrzut ekranu przedstawiający wygenerowany graf CPM:



Zrzut ekranu przedstawiający wygenerowany harmonogram Gantta:



Najważniejsze fragmenty kodu źródłowego

- Wyznaczenie najwcześniejszego czasu zaistnienia zdarzenia:

```
//wyznaczenie najwczesniejszego czasu zaistnienia zdarzenia
events.forEach((e, i, eventsArr) => {
  if (i === 0) return;
  var temp1: number = 0;
  actionsList.forEach((al) => {
    if (al.nextEventId === e.eventId) {
      if (events[al.prevEventId - 1].earliest + al.duration > temp1) {
        temp1 = events[al.prevEventId - 1].earliest + al.duration;
      }
    }
  });
  eventsArr[i].earliest = temp1;
});
```

- Wyznaczenie najpóźniejszego czasu zaistnienia zdarzenia:

```
//wyznaczenie najpozniejszego czasu zaistnienia zdarzenia
let index = 0;
for (let i = events.length - 1; i >= 0; i--) {
  if (i === events.length - 1 || i === 0) {
    events[i].latest = events[i].earliest;
  } else {
    let temp: Action = {
      name: "",
      duration: 1000000,
      nextEventId: 0,
      prevEventId: 0,
    };
    index = 0;
    actionsList.forEach((al) => {
      if (al.prevEventId === i + 1) {
        if (index === 0)
        {
          temp = al;
        }
        else
        {
          if (events[al.nextEventId - 1].latest - al.duration <events[temp.nextEventId - 1].latest - temp.duration)
          {
            temp = al;
          }
        }
        index++;
      }
    });
    events[i].latest = events[temp.nextEventId - 1].latest - temp.duration;
  }
}
```

- Wyznaczenie zapasu zdarzenia:

```
//zapas
events.forEach((e) => (e.stock = e.latest - e.earliest));
```

- Wyznaczenie ścieżki krytycznej:

```
//sciezka krtyczyna
const criticalActions: Action[] = [];
const criticalEvents: Event[] = [];
for (var i = 0; i < max; i++) {
  if (events[i].stock === 0) {
    criticalEvents.push(events[i]);
    for (var j = 0; j < actionsList.length; j++) {
      if (
        actionsList[j].prevEventId === events[i].eventId &&
        events[actionsList[j].nextEventId - 1].stock === 0
      ) {
        if (
          events[actionsList[j].nextEventId - 1].earliest -
            actionsList[j].duration ===
          events[actionsList[j].prevEventId - 1].earliest
        )
          criticalActions.push(actionsList[j]);
      }
    }
  }
}
```

- Wyznaczenie danych do narysowania grafu CPM

```
OutputData.events.forEach((e, index, arr) => {
  const data =
    e.earliest >= 10 || e.latest >= 10
      ? `${e.eventId}\n${e.earliest}  ${e.latest}\n${e.stock}`
      : `${e.eventId}\n${e.earliest}     ${e.latest}\n${e.stock}`;
  let color2 = "#000";
  let border = 1;
  if (
    OutputData.criticalPathEvents.find((cpa) => cpa.eventId === e.eventId)
  ) {
    color2 = "red";
  }

  if (e.eventId === 1 || e.eventId === OutputData.events.length) {
    border = 4;
  }
  cy.add({
    group: "nodes",
    data: {
      id: `${e.eventId}`,
      label: data,
      color2,
      border,
    },
  });
});
OutputData.actions.forEach((a) => {
  let color = "rgb(27, 91, 143)";
  if (OutputData.criticalPathActions.find((cpa) => cpa.name === a.name)) {
    color = "red";
  }

  cy.add({
    group: "edges",
    data: {
      id: a.name + a.duration,
      source: a.prevEventId,
      target: a.nextEventId,
      color,
    },
  });
});
```

- Wyznaczenie danych do harmonogramu Gantta:

```
export const dataGantt = (outputData: OutputData) => {
  const arrayGant: GanttData[] = [];
  let temp2 = 0;
  outputData.actions.forEach((a) => {
    //czas i pokrycie
    const today = new Date();
    const dataStart = new Date();
    let percentComplete: number;
    let dur: number;
    if (temp2 === 0) {
      dataStart.setDate(
        today.getDate() // + outputData.events[a.prevEventId].earliest
      );
      percentComplete = 100;
      dur = a.duration;
    } else {
      dataStart.setDate(
        today.getDate() + outputData.events[a.prevEventId - 1].earliest
      );
      percentComplete =
        (100 * a.duration) /
        (outputData.events[a.nextEventId - 1].latest -
          outputData.events[a.prevEventId - 1].earliest);
      dur =
        outputData.events[a.nextEventId - 1].latest -
        outputData.events[a.prevEventId - 1].earliest;
    }
    temp2++;

    //czy na sciezce krytycznej
    let criticalX = "No Critical";
    outputData.criticalPathActions.forEach((c) => {
      if (c.name === a.name) criticalX = "Critical";
    });
```

```
    //dependencies
    let temp = "";
    var counter: number = 0;
    outputData.actions.forEach((a1) => {
      if (a.prevEventId === a1.nextEventId) {
        if (counter === 0) temp += a1.name;
        else temp += "," + a1.name;
        counter++;
      }
    });
    var dependencies;
    if (temp === "") dependencies = null;
    else dependencies = temp;

    arrayGant.push({
      task_id: `${a.name}`,
      task_name: `${a.name}`,
      critical: criticalX,
      start_date: dataStart,
      end_date: null,
      duration: daysToMilliseconds(dur),
      percent_complete: percentComplete,
      dependencies,
    });
  });

  return arrayGant;
};
```