Zajęcia C-2: “Kolejka”

Cel zajęć i efekty uczenia

Główne cele zajęć / materiał do opanowania:

- Struktura kolejki, szablon *queue* z STL-a
- Pojęcie “abstrakcyjnej” struktury danych
- [opcjonalnie] Wstęp do techniki zwanej “obgryzaniem grafu”

Dodatkowe cele:

- Idea, cele i sensowność optymalizacji pamięciowej

Zadania do rozwiązania na sprawdzarce

Wojna
Dwoje graczy gra w wojnę, każde z własną częścią talii: w jednej turze gracze odkrywają swoje wierzchnie karty, gracz z wyższą kartą zabiera obie i dokłada na koniec swojej talii. Zasymulować podaną liczbę tur rozgrywki.

[opcjonalnie] ***Obrazy i pokoje***
Jest n pokojów, w każdym na początku stoi jedna osoba. W każdym pokoju na ścianie napisany jest numer pewnego innego pokoju. Na dźwięk dzwonka każdy przechodzi ze swojego pokoju do tego, którego numer widzi na ścianie. Wyznaczyć wszystkie pokoje, które nigdy nie staną się puste.

Plan zajęć

Szacunkowy czas trwania: 2 godziny lekcyjne (sama kolejka), lub 4 godziny (z techniką obgryzania).

1. Organizacja danych w postaci kolejki, struktura danych
    - *W kolejce potrzebujemy dopisywać nowe elementy, i wybierać pierwszy dodany element.*
    - *Zwykła tablica nie potrafi tego robić (choćby z uwagi na fakt, że ma stały rozmiar), za to potrafi np. podawać element na zadanej pozycji.*
    - *Różne sposoby uporządkowania danych - **struktury danych** - dobieramy w zależności od tego, czego aktualnie potrzebujemy.*
2. Kolejka w C++
    - *Podobnie jak wektor, kolejkę można napisać na różnych typach danych - typy takie, jak kolejka i wektor nazywają się **szablonami**.*
    - *Warto zwrócić uwagę, że front() tylko podgląda (nie zmieniając) pierwszy element kolejki, a pop() tylko go usuwa (nic nie zwracając).*
3. Własna implementacja kolejki

- *Zauważamy, że samo dokładanie na koniec można zrealizować znanym już typem wektora (jednak wciąż nie wiemy dokładnie, jak działa wektor!).*
- *Zabieranie elementy z początku kolejki - wskaźnik czoła (head) kolejki.*
- *Jak poradzić sobie z pisaniem kolejki bez wektora? Dwa wskaźniki: czoło i ogon kolejki. Ale co, jeśli będzie wiele operacji?*
- *Możliwość I: zadeklarować tablicę o rozmiarze takim, jak liczba operacji na kolejce. Wtedy ogon kolejki nigdy nie dojdzie do końca tablicy.*
- *Możliwość II: kolejka cykliczna, obie operacje modulo długość tablicy. Wtedy zużycie pamięci będzie niższe, ale kolejka zepsuje się, jeśli będzie na niej za dużo elementów.*

4. Której możliwości używać? Czy warto oszczędzać pamięć?
   - *Zawsze patrzymy, do czego będziemy używać kolejki. Jeśli będzie na niej naraz dużo elementów, nie warto robić kolejki cyklicznej. Jeśli jednak będzie dużo operacji, a naraz mało elementów, kolejka cykliczna jest najlepszym pomysłem.*
   - *Dobieranie struktury i implementacji do zadania to bardzo ważna część projektowania algorytmu.*
   - *Warto oszczędzać pamięć, ale nie należy z tym przesadzać - jeśli oszczędność nie jest paląca, a czyni program nieczytelnym, żmudnym do napisania, łatwym do pomylenia się, albo nieoptymalnym czasowo - nie oszczędzamy niepotrzebnie.*
5. Zadanie “Wojna”
   - *Jest to bardzo prosta implementacja kolejki - potrzebne są dwie zmienne tego typu, jedna na każdą talię. Implementacja to zwyczajna symulacja gry, tura po turze.*
   - *Kwestia techniczna: przed każdą turą koniecznie należy sprawdzać, czy kolejka jest pusta, żeby nie wykonać operacji wyciągania z pustej kolejki.*
   - *Operacje na pustej kolejce mogą mieć trudne do przewidzenia skutki, warto uczulić na to uczniów.*
6. Obgryzanie, zadanie “Obrazy i pokoje”
   - *Pokoje, do których nie ma wejścia (nic do nich nie kieruje), opróżnią się w pierwszej turze i już nigdy nie będą używane. Możemy więc je usunąć z gry.*
   - *W wyniku tego usunięcia pewne inne pokoje straciły wszystkie wejścia - one opróżnią się w drugiej turze. Możemy więc teraz z kolei je usunąć, a potem powtarzać tę operację tak długo, aż będzie możliwa.*
   - *Na końcu pozostanie sytuacja, w której wszystkie pokoje są pełne i do każdego jest wejście - pozostałe pokoje będą zatem już zawsze pełne.*
   - *Prosta implemetacja miałaby złożoność $O(n^2)$. Złożoność liniową uzyskujemy w następujący sposób:*
     - *dla każdego pokoju p pamiętamy in[p] - liczbę czynnych wejść do pokoju, na poczatku jest to liczba wszystkich wejść, którą liczymy podczas wczytywania danych;*
     - *pokoje, dla których in[p] = 0, wrzucamy na kolejkę;*
     - *ściągamy pokój p z kolejki, oznaczamy go jako opróżniony, po czym rozważamy pokój x, do którego prowadzi wyjście z p - ponieważ p się*

opróżnił, wejście przestaje być aktywne: zmniejszamy in[x] o 1. Jeśli in[x] jest teraz równe 0, dorzucamy x do kolejki;

- *powtarzamy powyższy krok, aż kolejka się nie opróżni;*
- *pokoje, które pozostaną, mają wartość in[] dodatnią, a więc już zawsze ktoś będzie do nich wchodził.*