Rozwiązania (każde na oddzielnej kartce) należy oddawać do mojej przegródki w sekretariacie Instytutu Matematyki do **22.11.2016**, godz. **9:50**. Oddana praca nie może mieć mieć więcej niż jednego autora. Podczas rozwiązywania można korzystać ze wszelkich pomocy naukowych.

W zadaniach należy opisać działanie algorytmu realizującego polecenie (nie trzeba pisać pseudokodu ale czasem może być to z różnych powodów wygodne).

Można zakładać, że dane są dobrane w taki sposób, że drzewa BST działają w czasie logarytmicznym (nie trzeba korzystać z AVL).

ZADANIE 1 (10p.)

Dany jest typ T wraz z porządkami liniowymi $\prec_1, \prec_2, \ldots, \prec_n$. Zaprojektować strukturę danych *Set* reprezentującą podzbiór elementów T wraz z funkcjami

- Init(Set s) – inicjalizuje zbiór s;
- Insert(Set s, T x) – dodaje element x do zbioru s;
- Boolean Find(Set s, T x) – sprawdza czy w zbiorze s znajduje się element x;
- Delete(Set s, T x) – usuwa element x;
- T ExtractMin(Set s, int k) – usuwa najmniejszy element względem porządku $\prec_k$;
- T Next(Set s, T x, int k) – zwraca najmniejszy (względem porządku $\prec_k$) element większy od x, jeśli element x jest maksymalnym elementem w s względem $\prec_k$ ma o tym poinformować (zwrócić wyjątek; można napisać np. Throw(x jest maksymalny))

Wszystkie funkcje powinny działać w czasie $\mathcal{O}(n \cdot \ln(|S|))$, gdzie $|S|$ jest liczbą elementów w zbiorze S.

ZADANIE 2 (10p.)

U lekarza występują dwa rodzaje pacjentów zwykli i uprzywilejowani. Przy tym zachowany jest zwyczaj, że grupy są umieszczane w oddzielnych kolejkach (odpowiednio q oraz q_p), a następnie w ramach każdej grupy wchodzą do gabinetu w kolejności przyjścia. Panuje zwyczaj, że pacjent uprzywilejowany wchodzi do lekarza szybciej niż gdyby nie był uprzywilejowany. Jednakże każdy pacjent powinien w końcu wejść do lekarza w związku z czym co jakiś czas pacjent uprzywilejowany powinien przepuścić (pierwszego w kolejce) pacjenta nieuprzywilejowanego (np. jeśli ten czekał długo).

(2p.) Opisz zasadę działania funkcji:
PrzyszedlZwykly(Struktura_kolejkowa s, Osoba x),
PrzyszedłUprzywilejowany(Struktura_kolejkowa s, Osoba x) oraz
Osoba NastępnyProszę(Struktura_kolejkowa s) w taki sposób aby czasy czekania nowo przybyłego pacjenta uprzywilejowanego wynosił $\mathcal{O}(|q_p|)$ natomiast pacjenta zwykłego $\mathcal{O}(|q|+|q_p|)$. ($|q|$ oraz $|q_p|$ oznaczają, odpowiednio liczbę pacjentów zwykłych oraz uprzywilejowanych w momencie PRZYJŚCIA nowego pacjenta).

(4p.) Niech $c(s)$ spełnia warunek $c(s)/s \to \infty$. Opisz zasadę działania funkcji jak w poprzednim podpunkcie tak aby czas czekania pacjenta uprzywilejowanego wynosił $|q_p|\,(1 + o(1))$, natomiast pacjenta zwykłego wynosił $\mathcal{O}(c(|q| + |q_p|))$.

(4p.) Pokazać, że jeśli czas czekania pacjenta uprzywilejowanego wynosi $|q_p|\,(1 + o(1))$, to czas oczekiwania pacjenta zwykłego nie może wynosić $\mathcal{O}(|q| + |q_p|)$

ZADANIE 3 (10p.)

Prolog: Mieszkańcy bajtocji używają dużo list dwukierunkowych wyposażonych w standardowe procedury

- Init(List q) – inicjuję listę q;
- push_back(List q, T x) – umieszcza element x na końcu listy q;
- push_front(List q, T x) – umieszcza element x na początku listy q;
- bool isempty(List q) – sprawdza czy lista q jest pusta;
- T pop_back(List q, T x) – zwraca ostatni element na liście q jednocześnie go usuwając;
- T pop_front(List q, T x) – zwraca pierwszy element na liście q jednocześnie go usuwając;
- Destroy(List q) – usuwa listę q;
- Destroyall() – usuwa wszystkie listy w bajtocji.

Dodatkowo, na wniosek mieszkańców król stworzył jeszcze jedną procedurę List clone(List q), która zwraca listę będącą kopią listy q (i od tej pory zachowują się one jak dwie niezależne listy). Procedura ta działa w czasie proporcjonalnym do długości kolejek jednakże mieszkańcy ochoczo z niej korzystali. Doprowadziło to do tego, że wszystkie listy były zajęte tworzeniem swoich kopii, miasto było pełne kolejek a w całym królestwie zapanował chaos...

Polecenie: Twoim zadaniem jest pomóc królowi i zaprojektować listę dwukierunkową wraz z procedurą clone. Wywołanie procedury clone nie powinno prowadzić do kopiowania wszystkich elementów na liście a jedynie sprawiać takie wrażenie z punku widzenia użytkownika. Lista powstała na skutek wywołania procedury clone powinna zachowywać się w sposób niezależny od swojego przodka – w szczególności operacje wywoływane na jednej z list nie mogą wpływać na zawartość pozostałych.

Aby zadanie zostało ocenione na maksymalną liczbę punków wszystkie funkcje poza Destroy powinny działać w czasie $\mathcal{O}(\log(n))$, gdzie n jest liczbą list w całym królestwie. Za rozwiązanie, w którym czas wszystkich operacji będzie większy, choć niezależny od ilości elementów na liście uzyskasz 6 punków. Za zapisanie rozwiązania dotychczas używanego w bajtocji – 2 punkty.

Uwaga: W zadaniu można założyć, że mamy do dyspozycji odpowiednik STL-owej struktury Map[1] z dodawaniem, usuwaniem, wyszukiwaniem oraz zmianą wartości na konkretnym argumencie w czasie logarytmicznym względem liczby elementów w mapie. Implementacja takiej struktury może być wykonana na drzewie AVL ale nie trzeba jej opisywać.

Uwaga 2: Możemy tworzyć dowolną liczbę danych widocznych globalnie dla całej bajtocji.

[1] szczegółowa dokumentacja znajduje się pod adresem `http://www.cplusplus.com/reference/map/map/`