Książka adresowo - społecznościowa

Realizacja zadania 3 z przedmiotu: podstawy programowania
Dokumentacja projektowa
Autor: Paweł Ostaszewski

Opis:

Program symulujący książkę adresowo społecznościową. Program daje możliwość przechowywania danych o osobach oraz informacji o kontaktach między nimi. Baza umożliwia wykonywanie rozmaitych operacji takich jak: wczytywanie bazy z pliku, zapis do pliku, dodawanie i usuwanie pozycji (osób), dodawanie informacji o znajomościach między osobami, sortowanie bazy względem różnych kryteriów, wypisywanie bazy na ekran, szukanie sposobu na nawiązanie kontaktu między osobami.

Implementacja:

Baza została zaimplementowana przy użyciu dynamicznych struktur danych. Baza jest dynamiczną listą węzłów, z których każdy zawiera informacje o jednej osobie oraz ma połączenie z dynamiczną listą krawędzi wychodzących z danego węzła. Krawędzie oznaczają tutaj znajomości między osobami. W ten sposób bazę można rozumieć jako graf powiązań między osobami. Program nie używa żadnych tablic identyfikatorów, ani tablic osób, a do każdego węzła odwołuje się przechodząc listę węzłów (w sposób sekwencyjny a nie poprzez dostęp bezpośredni). Czas dostępu do węzła może mieć złożoność O(liczba węzłów), ale za to w programie nie są potrzebne dynamiczne tablice i nie ma problemów z obsługą tych tablic, takich jak zwiększanie rozmiaru tablicy gdy stanie się pełna lub zmniejszanie rozmiaru gdy będzie zapełniona tylko w 5%. Nie ma też problemów związanych z dziurami w tablicach oraz przemieszczaniem pamięci wewnątrz tablicy.

Program wykorzystuje również masowo operacje na wskaźnikach oraz dynamiczną alokacje pamięci (przy użyciu funkcji malloc) oraz jej zwalnianie (przy użyciu free). Dzięki temu w programie nigdy nie są kopiowane dane z jednej struktury do drugiej. Poza tym program cały czas pracuje na tej samej bazie danych i nie wykonuje żadnych zbędnych i nadmiarowych alokacji pamięci. Każda funkcja w programie operująca na książce adresowej przyjmuje adres do tej struktury i odnosi się do jej elementów za pomocą wskaźników. Program jest też dobrze zabezpieczony przed wyciekami pamięci.

Funkcjonalność wyszukiwania sposobu na nawiązanie znajomości między osobami została zaimplementowana jako wyszukiwanie najkrótszej ścieżki w grafie powiązań między osobami przy użyciu algorytmu Dijkstry. W algorytmie Dijkstry wykorzystuje się kolejkę priorytetową do przechowywania najmniejszych odległości wszystkich węzłów od węzła źródłowego, która w programie została zaimplementowana jako kopiec binarny typu min, żeby uzyskać dobrą złożoność obliczeniową operacji na kolejce priorytetowej (lepszą niż w przypadku listy FIFO). Wszystkie operacje na kopcu binarnym w programie są analogiczne do tych opisanych w książce „Wprowadzenie do algorytmów” T. Cormena.

1. Struktury użyte w programie:

Adres (adres) zawierający następujące składowe:
- (char[32]) string zawierający nazwę ulicy
- (char[7]) kod pocztowy w formacie 01-234. Siódmy znak tablicy to ‘\0’
- (char[32]) string zawierający nazwę miasta zamieszkania
- (int) numer domu
- (int) numer mieszkania

Osoba (wezel) zawierająca następujące składowe:
Dane osobowe:
- (int) identyfikator
- (char[32]) pierwsze imię
- (char[32]) string zawierający drugie imię, użytkownik może podać drugie imię lub nie. Jeśli drugie imię nie zostanie podane to w stringu drugie_imie przechowujemy napis "_", co ułatwia operacje wczytywania i zapisywania do pliku.
- (char[32]) nazwisko
- (int) numer telefonu
Dane adresowe:
- (adres) adres zamieszkania (patrz wyżej)
Dane wykorzystywane do utrzymania struktury grafu:
- (wezel *nastepny) wskaźnik do następnego węzła w liście wszystkich węzłów grafu. Przechodząc listę wszystkich węzłów grafu używamy właśnie tego wskaźnika (wezelwsk = wezelwsk->nastepny).
- (krawedz *pierwszy) wskaźnik na pierwszą krawędź w liście wszystkich krawędzi wychodzących z danego węzła. Innymi słowy pierwsza znajomość w liście znajomych osób danej osoby (patrz niżej struktura Krawędź)
Dane wykorzystywane w algorytmie Dijkstry (patrz 5. Algorytm Dijkstry):
- (wezel *poprzednik) wskaźnik na poprzedni węzeł w najkrótszej ścieżce od węzła źródłowego do węzła docelowego.
Dzięki tej składowej można łatwo odtworzyć najkrótszą ścieżkę od węzła źródłowego do węzła docelowego. (patrz 8.4)
- (Int) dotychczasowa odległość między danym węzłem a węzłem źródłowym
- (int) liczba krawędzi dzielących dany węzeł od węzła źródłowego w najkrótszej ścieżce

Krawędź (krawedz) zawierająca następujące składowe:
- (wezel *cel) wskaźnik do węzła do którego prowadzi dana krawędź tzn. osoba z która ktoś zawarł znajomość
- (short waga) waga danej krawędzi, tzn. stopień znajomości w jakim osoba zawierająca daną krawędź wychodzącą zna osobę do której prowadzi dana krawędź (skala od 1 do 10)
- (krawedz *nastepny) wskaźnik na następną krawędź na liście krawędzi wychodzących z danego węzła tzn. następna znajomość danej osoby w liście wszystkich jej znajomości

Baza (graf) zawierające następujące składowe:
- (int) liczba elementów przechowywanych w bazie danych. Na początku jest równa 0;
- (int) bieżący identyfikator, który jest przypisywany do nowo dodanej osoby w bazie danych (na początku jest równy 1). Bieżący id zwiększa się za każdym razem gdy dodajemy nową osobę. Ściślej rzecz ujmując, po dodaniu nowej osoby wykonywane jest działanie biezacy_id = (biezacy_id+1) % INT_MAX, które powoduje zwiększenie bieżącego identyfikatora o 1 modulo maksymalna liczba całkowita jaką można zapisać na danej maszynie. Dzięki temu w bazie można dodawać i usuwać osoby teoretycznie nieskończenie wiele razy.
- (wezel*) wskaźnik do pierwszego węzła w liście wszystkich węzłów grafu, za pomocą tego wskaźnika możemy dostać się do każdej struktury w bazie.
W programie używane są dwie nazwy dla tej struktury: baza oraz graf (typedef baza graf)
Dzięki temu łatwiej jest zrozumieć kiedy posługujemy sie bazą jako grafem, a kiedy jako książką adresową.

Kopiec binarny typu min (kopiec_min) zawierający składowe:
- (wezel **tablica) tablica wskaźników do wszystkich węzłów grafu. Dzięki niej możemy łatwo się odwoływać do węzłów grafu w algorytmie Dijkstry. Nie są tworzone kopie węzłów ani nie są wykorzystywane tablice typu: int odległosc[n], wezel *poprzednik[n].
- (int) rozmiar tablicy tj. ilość węzłów w grafie

2. Menu główne

po uruchomieniu programu pokazuje się menu główne,
można w nim wybrać jedną z 11 opcji:
1. Dodawanie nowej osoby do książki adresowej
2. Usuwanie osoby z książki adresowej
3. Dodawanie znajomości między osobami
4. Usuwanie znajomości między osobami
5. Zmiana stopnia znajomości miedzy osobami
6. Wczytywanie bazy z pliku
7. Zapisywanie bazy do pliku
8. Wypisywanie książki adresowej na ekran
9. Wypisywanie sposobu na nawiązanie kontaktu miedzy osobami
10. Sortowanie książki adresowej
11. Koniec pracy programu

Menu główne pokazuje się za każdym razem po wykonaniu przez program jakiejś funkcjonalności,
aż do zakończenia pracy programu

3. Sortowanie

Sortowanie odbywa się na liście wszystkich węzłów grafu, tzn. zmienia się kolejność w jakiej węzły są ze sobą połączone. Sortowanie masowo wykorzystuje operacje na wskaźnikach. Jeśli jakaś para węzłów znajdują się w złej kolejności, tzn. pierwszy zawiera większą wartość niż drugi, to wskaźnik nastepny drugiego węzła pokazuje od tej pory na pierwszy węzeł. Wskaźnik nastepny pierwszego węzła zaczyna pokazywać na węzeł na który wcześniej pokazywał węzeł drugi. Węzeł pokazujący wcześniej na węzeł pierwszy pokazuje odtąd na węzeł drugi (za pomocą wskaźnika nastepny). Nie jest więc nigdzie kopiowana pamięć z jednego węzła do drugiego.
Do sortowania listy wszystkich węzłów grafu użyty został algorytm sortowania przez scalanie. Dzięki temu złożoność obliczeniowa sortowania to $O(n*\log_2(n))$, gdzie n to liczba wszystkich węzłów grafu. Ponadto złożoność pamięciowa to O(1) bo nie są tworzone żadne dodatkowe tablice n – elementowe, funkcja sortująca operuje na rzeczywistych węzłach grafu.
Sortowanie można wykonać według:
- identyfikatora
- imienia
- nazwiska

Wszystkich osób w grafie.

Ujmując ideę sortowania w sposób bardziej szczegółowy można powiedzieć, że funkcja sortowanie_przez_scalanie, przyjmuje jako argumenty: wskaźnik na pierwszy węzeł listy węzłów którą chcemy posortować, liczbę mówiącą jaka jest długość przesłanej listy oraz liczbę 1, 2 lub 3, która mówi, czy funkcja ma sortować odpowiednio według identyfikatora, imienia, czy nazwiska.
Funkcja sortowanie_przez_scalanie dzieli przesłaną listę mniej, więcej na pół przechodząc po niej wskaźnikiem wezelwsk po czym wywołuje się rekurencyjnie dla dwóch przepołowionych podlist. Na końcu funkcja scala dwie posortowane podlisty.
Funkcja scalanie_list działa w ten sposób, że przechodzi wskaźnikami wsk1 po pierwszej liście, wsk2 po drugiej. Jeśli pierwszy element pierwszej listy jest mniejszy od pierwszego elementu drugiej listy to wsk3->nastepny wskazuje na ten element, po czym wsk1, wsk3 przesuwają się do przodu za pomocą instrukcji wsk = wsk->nastepny. Analogicznie jest w przypadku gdy element pierwszej listy jest większy elementu drugiej listy. Jeśli któraś lista się wyczerpie to wsk3->nastepny wskazuje na pozostałość niewyczerpanej listy.
Funkcja scalanie_list zwraca wskaźnik do pierwszego węzła posortowanej, scalonej listy.

Na koniec można wspomnieć o tym, że sortowanie korzysta z funkcji string_compare, która sortuje napisy według porządku leksykograficznego, z tym, że ignoruje wielkość liter (inaczej niż strcmp). Dzięki temu użytkownik nie musi się przejmować formatem napisów, które wpisuje z klawiatury.
Sortowanie nazwisk wygląda np. w taki sposób:
Głowacki kowalski Nowicki, zamiast: Głowacki Nowicki kowalski

4. Wczytywanie danych z klawiatury

Wczytywanie wykonuje się za pomocą uogólnionej funkcji wczytywanie(), która przyjmuje następujące argumenty:
- (char *napis) string, który wypisuje na ekran za każdym razem gdy użytkownik wpisze niepoprawne dane
- bool (*kryterium)(char*) funkcja (kryterium), która sprawdza poprawność wpisanych przez użytkownika danych
- (char typ_danych) znak oznaczający typ wczytywanych danych: 'i' - integer 's' - string
- (void*) wskaźnik na zmienną typu int lub char* w której będą zapisane wczytane dane
Jeśli użytkownik wpisze niepoprawne dane to wypisany zostanie na ekran string zawarty w zmiennej napis oraz zostanie poproszony o ponowne wpisanie danych. Funkcja wczytująca występuje w programie wszędzie tam gdzie użytkownik wpisuje jakieś dane z klawiatury.

5. Algorytm Dijkstry

Wyszukiwanie najlepszego sposobu na nawiązanie znajomości między osobami jest zaimplementowane jako szukanie najkrótszej ścieżki w grafie między węzłem źródłowym i docelowym przy użyciu algorytmu Dijkstry. W algorytmie tym wykorzystuje się kolejkę priorytetową, która w programie została zaimplementowana jako kopiec binarny typu min. Dzięki temu mamy dobrą złożoność obliczeniową wielu operacji na kolejce priorytetowej. Np. operacja zmniejszająca wartość elementu (odległość od źródła) ma złożoność $O(\log_2 n)$ podczas gdy zmniejszenie wartości elementu w kolejce priorytetowej zaimplementowanej jako lista FIFO ma złożoność obliczeniową O(n).
Na początku działania algorytmu Dijkstry jest tworzony kopiec binarny typu min, zawierający odległości wszystkich węzłów od węzła źródłowego. Funkcja budowanie_kopca() alokuje pamięć na n - elementową tablicę (n – liczba węzłów w grafie) wskaźników do wszystkich węzłów grafu, za pomocą których kopiec binarny odwołuje się do węzłów. Funkcja budowanie_kopca() przypisuje następujące wartości do danych składowych wszystkich węzłów:
- Odległość od węzła źródłowego. Odległość węzła źródłowego od węzła źródłowego staje się równa 0, a odległość pozostałych węzłów staje się równa INT_MAX (w ten sposób oznaczamy że dany węzeł jest, póki co, nieosiągalny z węzła źródłowego).
- Poprzednik w najkrótszej ścieżce. Na początku dla każdego węzła do wskaźnika poprzednik jest przypisywana wartość NULL
- Liczba krawędzi dzieląca węzeł źródłowy od danego węzła. Na początku dana składowa liczba_krawedzi staje się równa 0 dla każdego węzła

Jeśli funkcja algorytm_dijkstry została wywołana z argumentem tryb równym 1 to funkcja wyszukuje najszybszą ścieżkę między węzłami (najmniejsza liczba pośredników między osobami). Jeśli została wywołana z argumentem tryb równym 2 to funkcja wyszukuje najskuteczniejszą ścieżkę między węzłami (poprzez osoby, które się najlepiej znają).

5.1. Wyszukiwanie najszybszej ścieżki

Wyszukiwanie najszybszej ścieżki polega na tym, że wyciągamy z kopca binarnego węzeł o najmniejszej odległości od źródła (złożoność O(1)) za pomocą funkcji pobierz_minimalny() na który wskazujemy wskaźnikiem o nazwie min. Następnie przeglądamy wszystkie jego krawędzie wychodzące. Jeśli odległość od źródła sąsiada węzła min jest większa niż 1+odległość od źródła węzła min to odległość sąsiada węzła min staje się równa 1+odległość od źródła węzła min. Ponadto zapisujemy, że poprzednikiem sąsiada węzła min jest węzeł min (wykorzystujemy w tym celu daną składową węzła poprzednik).

Wykonujemy operację wyciągania węzła o najmniejszej odległości od źródła i zmniejszania odległości jego sąsiadów aż do momentu gdy:
- dojdziemy do węzła docelowego
- przeszukamy całą spójną składową grafu zawierającą węzeł źródłowy tzn. węzeł docelowy jest nieosiągalny ze źródłowego (znajduje się poza spójną składową węzła źródłowego).

Wtedy zwalniana jest pamięć wykorzystywana przez kopiec binarny i jeśli węzeł docelowy jest osiągalny od węzła źródłowego to zwracany jest wskaźnik do węzła docelowego, dzięki któremu możemy już odtworzyć najszybszą ścieżkę za pomocą danej składowej węzła poprzednik. Jeśli węzeł docelowy nie jest osiągalny od węzła źródłowego to zwracany jest NULL.

5.2. Wyszukiwanie najskuteczniejszej ścieżki
Do określenia odległości danego węzła od węzła źródłowego wykorzystujemy następujący wzór

$$odległość = \sum_{k=1}^{n} \left(11 - w_k\right) * k$$

gdzie n to liczba krawędzi dzielących dany węzeł od węzła źródłowego, w_k to waga krawędzi k - tej (waga od 1 do 10). W ten sposób obliczamy na przykład, że jeśli osoba 1 zna osobę 2 w stopniu 10, osoba 2 zna osobę 3 w stopniu 10, osoba 3 zna osobę 4 w stopniu 10, a osoba 1 zna osobę 4 w stopniu 1 to wychodzi, że skuteczniejsza droga to 1, 2, 3, 4, a nie 1, 4 bo mamy 1*1 + 1*2 + 1*3 = 6 < 10 = 10*1.

Wyszukiwanie najskuteczniejszej ścieżki również polega na ciągłym pobieraniu z kopca binarnego węzła o najmniejszej odległości od źródła (węzeł wskazywany przez wskaźnik min) i przeglądaniu odległości od źródła jego sąsiadów. Uwaga: węzeł raz wyciągnięty z kopca binarnego już do niego nie wraca.
Jeśli odległość od źródła sąsiada węzła min jest większa niż odległość węzła min od źródła + $(11 – w_k)*k$
to odległość sąsiada węzła min staje się równa odległość od źródła węzła min od źródła + $(11 – w_k)*k$,
gdzie w_k to waga krawędzi między węzłem min a jego sąsiadem, a k to ilość krawędzi między sąsiadem węzła min a węzłem źródłowym.
Ponadto zapisujemy, że poprzednikiem sąsiada węzła min jest węzeł min (wykorzystujemy w tym celu daną składową węzła poprzednik).
Wyciąganie węzła o najmniejszej odległości od źródła i zmniejszanie odległości jego sąsiadów od źródła jest wykonywane aż do momentu gdy:
- przejrzymy wszystkie węzły grafu, czyli kopiec binarny stał się pusty. Taka sytuacja zachodzi gdy cały graf jest spójny.
- przeszukamy całą spójną składową grafu zawierającą węzeł źródłowy tzn. węzeł docelowy jest nieosiągalny ze źródłowego (znajduje się poza spójną składową węzła źródłowego).

Wtedy zwalniana jest pamięć wykorzystywana przez kopiec binarny i jeśli węzeł docelowy jest osiągalny od węzła źródłowego to zwracany jest wskaźnik do węzła docelowego, dzięki któremu możemy już odtworzyć najszybszą ścieżkę za pomocą danej składowej węzła poprzednik. Jeśli węzeł docelowy nie jest osiągalny od węzła źródłowego to zwracany jest NULL.

Na koniec warto zanotować, że zmniejszanie odległości od źródła przeglądanych węzłów wykonuje się za pomocą funkcji zmniejsz_odleglosc(), która pozwala na zmniejszenie odległości danego węzła od źródła przy zachowaniu struktury kopca binarnego typu min, tzn. każdy element (węzeł) w kopcu ma mniejszą odległość niż wszystkie elementy lewego poddrzewa i prawego poddrzewa.
Wszystkie operacje wykonywane na kopcu binarnym typu min są analogiczne do tych opisanych w książce Cormena (rozdział 6. Heapsort).

6. Operacje wejścia, wyjścia z użyciem plików

Baza jest przechowywana w plikach tekstowych o nazwach:
- przykladowa_baza.txt. Przykładowa baza zawierająca 50 osób dostarczona razem z programem
- ksiazka_adresowa.txt. Dowolna baza, która może zostać utworzona przez użytkownika przy użyciu funkcji zapisywanie_bazy().

Baza jest przechowywana w pliku w czytelnym dla użytkownika formacie, tak żeby było wiadomo jakie osoby znajdują się w bazie i jakie są powiązania między nimi, nie znając w ogóle kodu źródłowego programu.

Format pliku zawierającego bazę:
Na początku znajdują się główne informacje o książce adresowej: ilość elementów, bieżący identyfikator (patrz 1. Struktury).

Ksiazka adresowo-spolecznosciowa
Liczba elementow: 50, biezacy id: 51

Dalej znajdują się informacje o wszystkich osobach znajdujących się w bazie, identyfikator w bazie, dane osobowe i adresowe zapisane w następującym formacie:

Osoba, id 1
Dane osobowe:
Jan Bartosz Nowicki nr telefonu: 123456789
Adres:
Ulica Polna 17/2, kod pocztowy: 01-123 miasto: Warszawa

Na końcu znajdują się informacje o znajomościach między osobami zapisane w formacie:

Znajomi osoby o identyfikatorze 17:
Id 7 Jan Kowalski stopien znajomosci: 7
Id 8 Michal Szczesny stopien znajomosci: 9
Id 9 Andrzej Kulewicz stopien znajomosci: 8

6.1. Wczytywanie bazy z pliku

Funkcja wczytywanie_bazy() pyta się na początku użytkownika, czy baza ma być wczytana z pliku przykladowa_baza.txt, czy z pliku ksiazka_adresowa.txt. Jeśli plik nie został odnaleziony przez program to funkcja wypisuje odpowiedni komunikat i kończy działanie (Uwaga: plik zawierający bazę powinien znajdować się w tym samym katalogu co program). Jeśli plik został odnaleziony to program otwiera go i odwołuje się do niego za pomocą wskaźnika na typ FILE o nazwie plik. Po otwarciu pliku program oczyszcza bazę z poprzednich danych, jeśli jakieś były, za pomocą funkcji usuwanie_wszystkich_wezlow() (patrz 7. Operacje na grafie) i inicjalizuje bazę na nowo za pomocą funkcji inicjalizacja_bazy().
Następnie funkcja wczytuje dane z pliku linia po linii za pomocą funkcji, fscanf() i fgets().
Najpierw są wczytywane główne informacje, ilość elementów i bieżący identyfikator.
Potem w pętli są wczytywane po kolei wszystkie osoby z bazy aż do momentu gdy następna wczytana linia z pliku będzie inna niż linia „*Osoba, id 1”.*
Przy każdej iteracji pętli alokowana jest pamięć na nowy węzeł, przepisywane są dane osoby z pliku do węzła i tworzone jest powiązanie (odtwarzanie listy węzłów) między danym węzłem a poprzednikiem. Jeśli dany węzeł to pierwszy węzeł w grafie to zmienna składowa struktury baza - wskaźnik zrodlo wskazuje od tej pory na ten węzeł.
Po wczytaniu wszystkich osób z pliku wczytywane są znajomości między osobami (krawędzie).
Dla każdego węzła w liście węzłów wykonywana jest pętla while, która pozwala wczytać informacje o wszystkich znajomościach danej osoby.
W każdej iteracji pętli while alokowana jest pamięć na kolejną krawędź, przepisywana jest jej waga z pliku do danej składowej krawędzi waga i tworzone jest powiązanie (odtwarzanie listy krawędzi wychodzących z danego węzła) między daną krawędzią a poprzednikiem (analogicznie jak przy odtwarzaniu listy węzłów).

6.2. Zapisywanie bazy do pliku
Zapisywanie do pliku jest operacją odwrotną do wczytywania, przy jednoczesnym zachowaniu formatu pliku. Funkcja zapisywanie_bazy() nie pyta już użytkownika, z którego pliku ma korzystać, tylko od razu otwiera plik do zapisywania o nazwie ksiazka_adresowa.txt. Jeśli plik ten nie istniał wcześniej to jest tworzony nowy. Jeśli istniał przed uruchomieniem funkcji to jego zawartość jest usuwana i wpisywane są do niego nowe informacje. Funkcja odwołuje się do pliku za pomocą wskaźnika na typ FILE o nazwie plik tak samo jak w przypadku funkcji zapisywanie_bazy().
Wszystkie dane są zapisywane w takiej samej kolejności w jakiej są wczytywane. Najpierw główne informacje o bazie. Potem w pętli while informacje o wszystkich osobach, na końcu w pętli while informacje o krawędziach.

7. Operacje na grafie

Program posiada swego rodzaju bibliotekę umożliwiającą wykonywanie operacji na grafie, z której korzystają funkcje wykonujące operacje na książce adresowej (patrz niżej 8. Operacje na książce adresowej).

7.1 dodawanie węzła do grafu
Funkcja dodawanie_wezla() alokuje pamięć na nowy węzeł i przypisuje następujące wartości jego danym składowym:
- id staje się równy identyfikatorowi podanego jako argument funkcji
- nastepny – wskaźnik na następny węzeł w liście wszystkich węzłów grafu staje się równy NULL bo dodajemy nowy węzeł na koniec istniejącej listy węzłów grafu (za pomocą NULL oznaczamy koniec listy)
- pierwszy – wskaźnik na pierwszą krawędź w liście krawędzi wychodzących staje się równy NULL
Po zainicjalizowaniu węzła funkcja przebiega listę węzłów grafu i umieszcza węzeł na końcu (odpowiednio zmieniając wskaźniki).

7.2 dodawanie krawędzi między węzłami
Funkcja dodawanie_krawedzi() dodaje jedną krawędź wychodzącą od węzła pierwszego do węzła drugiego oraz jedną krawędź wychodzącą od węzła drugiego do węzła pierwszego (do list krawędzi wychodzących danych węzłów).
Na początku sprawdza czy podane wagi krawędzi mieszczą się przedziałach od 1 do 10. Dwie wagi dla każdej z nich ponieważ waga krawędzi od pierwszej osoby do drugiej jest różna od wagi krawędzi od drugiej osoby do pierwszej zgodnie ze specyfikacją zadania. Jeśli wagi nie mieszczą się w przedziałach od 1 do 10 to funkcja zwraca -1.
Następnie funkcja alokuje pamięć na dwie nowe krawędzie i przypisuje odpowiednie wartości do ich zmiennych składowych. NULL do wskaźnika nastepny (pokazujący na następną krawędź w liście), bo funkcja umieszcza nową krawędź na końcu listy krawędzi wychodzących każdego węzła (za pomocą NULL oznaczamy koniec listy)
Funkcja przechodzi przez listę obecnych krawędzi i umieszcza nową krawędź na końcu (odpowiednio zmieniając wskaźniki), przy okazji sprawdzając, czy dana krawędź prowadząca od danej osoby do drugiej nie została dodana już wcześniej. Jeśli została dodana już wcześniej to funkcja zwraca -2.

7.3 usuwanie wszystkich krawędzi wychodzących z danego węzła
Rekurencyjna funkcja usuwanie_krawedzi_wychodzacych() działa w ten sposób, że wywołuje się rekurencyjnie dla następnej krawędzi w liście krawędzi wychodzących, po czym zwalnia pamięć przydzieloną na krawędź na danym poziomie rekurencji. W ten sposób najpierw zwalniana jest krawędź ostatnia, potem przedostatnia, a na końcu pierwsza.
Gdyby próbować zwolnić najpierw pierwszą krawędź to zostałby zgubiony wskaźnik do następnych krawędzi na liście co spowodowałoby wyciek pamięci, dlatego funkcja zwalnia krawędzie od końca.

7.4 usuwanie krawędzi wchodzącej
Funkcja usuwanie_krawedzi_wchodzacej() działa w ten sposób, że dla dwóch węzłów podanych jako argument sprawdza, czy istnieje krawędź „wchodząca" od węzła źródłowego do węzła docelowego. Jeśli istnieje to jest ona usuwana (jest zwalniana pamięć dla danej krawędzi i przestawiane są wskaźniki). Jeśli krawędź nie istnieje to funkcja nie robi nic.

7.5 usuwanie węzła
Funkcja usuwanie_wezla() zwalnia pamięć przydzieloną na dany węzeł i usuwa wszystkie krawędzie wchodzące do danego węzła i wychodzące z danego węzła. Wykorzystywane są w tym celu funkcje usuwanie_krawedzi_wchodzacej() i usuwanie_krawedzi_wychodzacych(). Funkcja działa w ten sposób, że przechodzi listę wszystkich węzłów grafu i jeśli dany węzeł nie jest węzłem usuwanym to wywoływana jest dla niego funkcja usuwanie_krawedzi_wchodzacej() do węzła usuwanego. Jeśli dany węzeł jest usuwanym węzłem to wywoływana jest funkcja usuwanie_krawedzi_wychodzacych(), zwalniana jest pamięć przydzielona na ten węzeł i przestawiane są odpowiednio wskaźniki.

7.6 usuwanie wszystkich węzłów grafu
Rekurencyjna funkcja usuwanie_wszystkich_wezlow() jest wykorzystywana przy zwalnianiu pamięci całego grafu. Działa ona analogicznie do funkcji usuwanie_krawedzi_wychodzacych(), tzn. wywołuje się rekurencyjnie dla następnego węzła w liście wszystkich węzłów, po czym wywołuje funkcję usuwanie_krawedzi_wychodzacych() oraz zwalnia pamięć przydzieloną na węzeł w danym poziomie rekurencji. W ten sposób najpierw zwalniany jest węzeł ostatni, potem przedostatni, a na końcu pierwszy.

8. Operacje na książce adresowej

Wszystkie funkcje wykonujące operacje na bazie odwołują się do odpowiednich funkcji operujących na grafie (patrz 7. Operacje na grafie).

8.1 dodawanie nowej osoby do bazy
Funkcja dodawanie_osoby() korzysta z odpowiedniej funkcji operującej na grafie dodawanie_wezla().
W trakcie dodawania nowej pozycji użytkownik podaje kolejne atrybuty nowej osoby,
które są zapisywane w zmiennych lokalnych funkcji.
Jeśli użytkownik poda błędne dane to program wypisze odpowiedni komunikat
i poprosi o ponowne wpisanie danych.
Po wczytaniu wszystkich danych sprawdzane jest czy pozycja o danym imieniu i nazwisku nie istnieje już w bazie. Jeśli istnieje to do istniejącego węzła w grafie są przepisywane dane o adresie i telefonie ze zmiennych lokalnych funkcji.
Jeśli osoba o danym imieniu i nazwisku nie istnieje w bazie to:
- liczba_elementow zwiększa się o 1
- tworzony jest nowy węzeł za pomocą funkcji dodawanie_wezla(), któremu przypisywany jest identyfikator biezacy_id
- bieżący identyfikator bazy jest zwiększany o 1 za pomocą operacji biezacy_id = (biezacy_id+1) % INT_MAX (patrz 1. Struktury)
- przepisywane są dane osobie ze zmiennych lokalnych funkcji do danych składowych nowego węzła w grafie

8.2 usuwanie osoby z bazy
Funkcja usuwanie_osoby() korzysta z odpowiedniej funkcji operującej na grafie usuwanie_wezla().
Funkcja na początku wypytuje użytkownika o identyfikator osoby, którą chce usunąć.
Program przechodzi wtedy listę węzłów grafu w poszukiwaniu węzła o podanym identyfikatorze.
Jeśli dany węzeł nie występuje w grafie
to program wypisuje odpowiedni komunikat i wraca do menu głównego.
Jeśli pozycja o danym id występuje w bazie to:
- wywoływana jest funkcja usuwanie_wezla()
- liczba_elementow zmniejsza się o 1

8.3 wypisywanie bazy na ekran
Funkcja wypisywanie_bazy() wypisuje na ekran dane wszystkich osób istniejących w bazie oraz relacje pomiędzy nimi.
Funkcja na początku sortuje bazę względem nazwisk (aby wypisywać bazę bez uprzedniego sortowania po nazwiskach trzeba ująć linijki 1111 i 1113 programu w komentarz).
Funkcja przechodzi po wszystkich węzłach w liście węzłów grafu i wypisuje ich dane osobowe, dane adresowe oraz ich relacje z innymi osobami w bazie.
Dane o każdej osobie są wypisywane na ekran w następującym formacie:

Id = 4
Imiona: Boguslaw Przemyslaw, Nazwisko: Nowacki
Adres:
ulica Sokratesa 13/12, kod pocztowy: 01-248, miasto: Krakow
nr telefonu: 630573512

Znajomi osoby:
id 3, Jan Kmicic, stopien znajomosci: 1
id 5, Jakub Jakubowski, stopien znajomosci: 2
id 30, Tomasz Wozniak, stopien znajomosci: 8

8.4 wyszukiwanie sposobu na nawiązanie kontaktu między dwoma osobami
Funkcja najkrotsza_sciezka(), która realizuje daną funkcjonalność pyta użytkownika o to, czy chce wyszukać najszybszą ścieżkę (najmniejsza liczba pośredników) czy najskuteczniejszą ścieżkę (poprzez osoby, które się najbardziej znają). Funkcja pyta też o identyfikatory osób między którymi ma być znaleziona ścieżka. Jeśli w bazie nie istnieje osoba o podanym identyfikatorze to funkcja wypisuje odpowiedni komunikat i wraca do menu głównego. Jeśli identyfikator pierwszej osoby jest równy identyfikatorowi drugiej to nie ma sensu szukać między nimi ścieżki, więc funkcja wypisuje odpowiedni komunikat i wraca do menu głównego.
Następnie funkcja wywołuje funkcję algorytm_dijkstry() z argumentem tryb równym 1 jeśli szukamy najszybszej ścieżki lub równym 2 jeśli szukamy najskuteczniejszej.
Jeśli istnieje ścieżka między danymi osobami to jest ona wypisywana jest na ekran za pomocą rekurencyjnej funkcji wypisywanie_najkrotszej_sciezki(). Funkcja ta przyjmuje jako argument wskaźnik do węzła docelowego w najkrótszej ścieżce i wywołuje się rekurencyjnie dla węzła, który jest poprzednikiem danego węzła w danej ścieżce. Następnie wypisywane są na ekran informacje o węźle na danym poziomie rekurencji. W ten sposób wywołania są wykonywane od docelowego węzła do źródłowego, a wypisywanie na ekran od węzła źródłowego do docelowego.

9. Pomiar czasu wykonywania funkcjonalności

Dla każdej funkcjonalności w programie mierzone jest, ile czasu wykonywała się dana funkcjonalność (przy użyciu struktur i funkcji z biblioteki standardowej time.h). W tym celu dla każdej funkcjonalności są deklarowane dwie zmienne typu clock_t poczatek i koniec. Na początku wykonywania się danej funkcjonalności wykonywana jest instrukcja poczatek = clock() (clock() zwraca ilość cykli zegara od początku działania programu), a na końcu koniec = clock(), po czym na ekran wypisywana jest zmiennoprzecinkowa wartość (double)(koniec-poczatek)/CLOCKS_PER_SEC. ClOCKS_PER_SEC jest stałą, która mówi ile wykonuje się cykli zegara w ciągu sekundy na danym komputerze. Dzięki temu wiemy ile sekund wykonywała się dana funkcjonalność.
Niestety w programie niektóre funkcjonalności wykonują się tak szybko, że na ekran wypisywana jest wartość 0.0000000. Czas wykonywana funkcjonalności dałoby się dobrze zmierzyć gdyby w bazie znajdowało się kilka tysięcy pozycji. Niestety przykładowa baza zawiera tylko 50 pozycji.