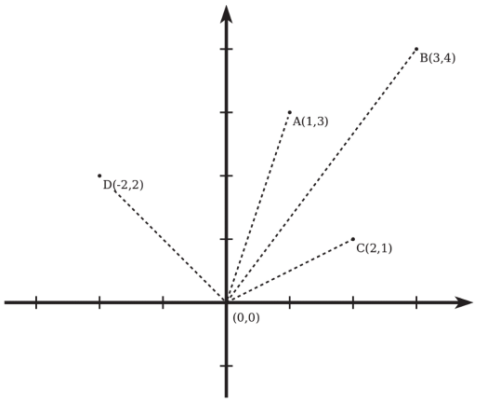
1 Zasady oceniania

1. 75% zaliczonych prac na zaliczenie trymestru
2. Próg zaliczenia jednej pracy nie wyższy niż 51%
3. Prace na zaliczenie będą na poziomie maturalnym

2 Zadanie algorytmiczne – szczyty

W pseudokodzie, lub wybranym przez siebie języku programowania, zapisz algorytm, który rozwiązuje następujący problem: na wejściu dane są niepusta lista współrzędnych (szczytów górskich) i długość tej listy, a w punkcie (0, 0) ustawiony jest obserwator. Algorytm powinien znaleźć szczyt, który dla obserwatora jest najbardziej po prawej stronie. Możesz założyć, że wszystkie szczyty mają dodatnią współrzędną y. Dalsze szczyty są przysłaniane przez bliższe, jeżeli leżą w tej samej linii. Dla przykładowego rysunku poniżej wynikiem działania algorytmu są współrzędne punktu C.



Proponowane rozwiązanie

W proponowanym rozwiązaniu skorzystamy ze wzoru na tangens kąta nachylenia prostej do osi OX. Proponowane rozwiązanie zakłada, że $x \neq 0$, więc wymaga jeszcze drobnej modyfikacji, by było w pełni poprawne.

```
procedure SZCZYTY(list, N)
    res = list[0]
    for (x, y) in list do
        (rx, ry) = res
        tg_res = ry/rx
        tg = y/x
        if tg · tg_res < 0 then
            if tg > 0 then
                res = (x, y)
            end if
        else
            if tg < tg_res then
                res = (x, y)
            else if tg == tg_res AND x < rx then
                res = (x, y)
            end if
        end if
```

```
    end for
     return res
end procedure
```

3 SQL

SQL to język służący do budowania zapytań do baz danych. Istnieją różne warianty języka, np. MySQL, PostgresQL, OracleSQL, MSSQL i wiele innych, które różnią się nieznacznie między sobą, ale główne zasady działania pozostają takie same.

Dane w bazach danych są zebrane w tabele. Tabele składają się z kolumn o predefiniowanych typach, a dane przechowywane są jako wiersze tabeli. Tabela powinna mieć też kolumnę będącą kluczem głównym (`UNIQUE_ID`), które nie może przyjąć dwa razy tej samej wartości (baza danych tego pilnuje). Zazwyczaj na takim polu ustawia się własność `AUTO_INCREMENT`, która automatycznie nadaje ID jako kolejne liczby naturalne. Niektóre z typów dostępnych w bazach to: `INT, FLOAT, DOUBLE, VARCHAR, TEXT, DATE, DATETIME` i inne. Istnieje również specjalny typ `REFERENCE`, który pozwala stworzyć odniesienie do wiersza innej tabeli. Przy próbie usunięcia wiersza, do którego istnieją odniesienia podniesiony zostanie wyjątek (przydatne w praktyce).

Podstawowe operacje w bazie danych to: `SELECT, CREATE, UPDATE, SET, DELETE, DROP`. Na maturze w części teoretycznej wystarczy operacja `SELECT`.

Poniżej przedstawiono dwie przykładowe tabele:

`emp`:

id	name	position	sal	manager	dept
1367	"Marek"	"Operator serwera"	1200	1369	1
1368	"Agata"	"Programista"	1400	1369	1
1369	"Aneta"	"Manager zespołu"	2500	NULL	1
1370	"Zbigniew"	"Dział Kadr"	1500	NULL	2

`dept`:

id	str	city
0	"Warszawska 14"	"Poznań"
1	"Startowa 9"	"Warszawa"
2	"Zabłocie 15"	"Kraków"

Podstawowa składnia operacji `SELECT` to `SELECT ... FROM ... WHERE`. Przykładowo zapytanie

```
SELECT * FROM emp
```

zwróci całą tabelę `emp`.

Jeśli interesują nas tylko imiona pracowników zarabiających powyżej 1200, to możemy wykonać zapytanie:

```
SELECT id, name FROM emp WHERE sal > 1200
```

Uwaga: Wielkość liter w zapytaniach nie ma znaczenia, ale dla wygody warto pisać słowa kluczowe wielkimi literami, a pozostałe elementy małymi.

Wyniki możemy zapytań możemy sortować. Domyślne sortowanie jest rosnące i nie wymaga żadnych dodatkowych słów kluczowych. Sortowanie malejące wymaga dodatkowego słowa jak w przykładzie:

```
SELECT id, name FROM emp WHERE sal > 1200 ORDER BY name DESCENDING
```

Na typach dostępnych w bazie danych zdefiniowane są sensowne porządki (np. w tym wypadku leksykograficzny).

Najbardziej skomplikowaną operacją jest łączenie tabel przy pomocy klauzuli `JOIN`. Przykładowo, jeśli chcemy wypisać imiona pracowników wraz z miastem, w którym są zatrudnieni, to możemy to zrobić jak poniżej. Dodatkowo zastosowaliśmy dodatkowe słowo kluczowe `AS`, które pozwala nam aliasować tabelę i skrócić zapis.

```
SELECT a.id, a.name, b.city
FROM emp AS a
WHERE sal > 1200
JOIN dept AS b ON a.dept = b.id
```

Zauważmy, że jeżeli np. tabela po lewej stronie (`emp`) nie będzie miała odpowiednika w tabeli po prawej stornie (`dept`), to ten wiersz pierwszej tabeli nie pojawi się w wyniku – a może chcemy, żeby pojawił się z jakimiś NULLami. Z tego powodu klauzula `JOIN` może występować w kilku wariantach:

- `LEFT JOIN` – wypisuje wszystkie elementy lewej tabeli i dopasowuje do nich elementy z prawej tabeli
- `RIGHT JOIN` – podobnie jak poprzednio, tylko w drugą stronę
- `FULL OUTER JOIN` – `LEFT JOIN` i `RIGHT JOIN` jednocześnie
- `NATURAL JOIN` – próbuje się domyślić po których kolumnach połączyć tabele – różnie to wychodzi, więc lepiej nie używać

Dostępne są też funkcje agregujące, np. `SUM, COUNT, MIN, MAX, AVERAGE` i inne. Korzystanie z nich wymaga dodania klauzuli `GROUP BY`, np. jeśli chcemy policzyć ilu jest pracowników o danym imieniu, to możemy zrobić to w następujący sposób:

```
SELECT a.name, COUNT(a.name) FROM emp AS a GROUP BY a.name
```

Przy korzystaniu z funkcji agregujących należy też pamiętać, że nie działa z nimi klauzula `WHERE`. Klauzula `WHERE` filtruje wiersze, więc nie możemy zbudować warunku z funkcją agregującą, ponieważ takie pole w danym wierszu nie istnieje.

Błędne jest zapytanie:

```
SELECT b.dept, SUM(a.sal)
FROM dept
JOIN emp ON a.dept = b.dept
WHERE SUM(a.sal) > 4000
GROUP BY b.dept
```

Zamiast tego powinniśmy napisać:

```
SELECT b.dept, SUM(a.sal)
FROM dept
JOIN emp ON a.dept = b.dept
HAVING SUM(a.sal) > 4000
GROUP BY b.dept
```

3.1 Zadanie

W bazie danych pewnego sklepu ze sprzętem komputerowym znajdują się tabele `klienci` i `zamowienia`, które przedstawiono poniżej. Dla tabel podano też kilka pierwszych wierszy.

imie	nazwisko	nr_telefonu
Weronika	Abacka	192 102 947
Jakub	Babacki	938 192 728
Szymon	Cabacki	722 719 382
...	...	...

nr_zamowienia	produkt	ilosc	nr_telefonu
1	"Kabel USB"	2	938 192 728
2	"Klawiatura Bluetooth"	1	722 719 382
3	"Baterie AAA"	12	938 192 728
...	...	...	...

Napisz zapytanie w języku SQL, które wygeneruje listę wszystkich klientów, którzy złożyli mniej niż 3, lub nie złożyli żadnego zamówienia. Posortuj listę rosnąco po liczbie zamówień. Możesz założyć, że klienci mają różne numery telefonów, a numer zamówienia jest kluczem głównym w swojej tabeli.

Przykładowe rozwiązanie

```
SELECT a.nazwisko, a.nr_telefonu, COUNT(b.nr_zamowienia)
FROM klienci AS a
LEFT JOIN zamowienia AS b ON a.nr_telefonu = b.nr_telefonu
GROUP BY a.nr_telefonu
HAVING COUNT(b.nr_zamowienia) < 3
```

4 Następnym razem

Rzeczy do matury:

- MS Access – zadanie maturalne
- Proszę o zapoznanie się z wyciągiem z podstawy programowej zamieszczonym na stronie i sygnał, co jeszcze nie było przerabiane