Ćwiczenia z optymalizacji z użyciem klastrów komputerowych

Dla wszystkich zadań opracuj algorytm równoległy, a następnie zaimplementuj go w języku C++ z wykorzystaniem biblioteki MPI. Ocenie będą podlegały: poprawność wyniku, przyspieszenie oszacowane za pomocą trzech wykonań (dla 2, 4 i 6 procesów), a dopiero na końcu czas działania. Na przykład, program z czasami działania, odpowiednio, 120s, 60s, 45s będzie wyżej oceniony niż program działający w czasie, odpowiednio, 14s, 12s, 11s.

Ćwiczenie 1. Dana jest funkcja f(i), której dziedziną są liczby całkowite dodatnie. Chcemy się dowiedzieć dla ilu spośród liczb całkowitych j od 1 do 2 000 000 zachodzi własność: f(j)==c, gdzie liczba całkowita c jest zadana przez użytkownika jako parametr wywołania programu. Badaną funkcję zdefiniowano następująco:

```
int f ( int i ) {
  int i4_huge = 2147483647;
  int j, k, value = i;
  for ( j = 1; j <= 5; j++ ) {
    k = value / 127773;
    value = 16807 * ( value - k * 127773 ) - k * 2836;
    if ( value <= 0 )
       value = value + i4_huge;
  }
  return value;
}</pre>
```

Ćwiczenie 2. Gra w życie toczy się na skończonej planszy (płaszczyźnie) podzielonej na kwadratowe komórki. Każda komórka ma ośmiu "sąsiadów", czyli komórki przylegające do niej bokami i rogami. Każda komórka może znajdować się w jednym z dwóch stanów: może być albo "żywa" (włączona), albo "martwa" (wyłączona). Stany komórek zmieniają się krokowo. Stan wszystkich komórek w danym kroku jest używany do obliczenia stanu wszystkich komórek w kroku następnym. Po obliczeniu wszystkie komórki zmieniają swój stan dokładnie w tym samym momencie. Stan komórki zależy tylko od liczby jej żywych sąsiadów. W grze tej nie ma graczy w dosłownym tego słowa znaczeniu. Udział człowieka sprowadza się jedynie do ustalenia stanu początkowego komórek.

Zdefiniowano kilka wzorców reguł generowania, najbardziej rozpowszechnione są reguły wymyślone przez Conwaya (znanego brytyjskiego matematyka). W niniejszym zadaniu jednak przyjmiemy trochę inne reguły.

- Martwa komórka, która ma 1 lub 2 żywych sąsiadów, staje się żywa w następnym kroku (rodzi się).
- Żywa komórka z 1 albo 2 żywymi sąsiadami pozostaje nadal żywa; przy innej liczbie sąsiadów umiera (z "samotności" albo "zatłoczenia").

Napisz program równoległy (podobnie jak w przypadku innych ćwiczeń w języku C++ oraz modelu z przesyłaniem wiadomości) symulujący dwuwymiarowy binarny automat komórkowy do gry w życie. Przyjmij, że gra rozgrywa się w macierzy 8000 wierszy na 8000 kolumn, która na początku (w kroku 0) ma włączone komórki na obu przekątnych, a pozostałe ma wyłączone. Jaka będzie liczba włączonych komórek w kroku 400?

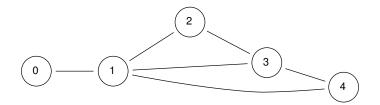
Ćwiczenie 3. W zadaniu tym będziemy rozważali graf nieskierowany, w którym wierzchołki ponumerowano kolejnymi liczbami całkowitymi zaczynając od zera. Graf ten nie będzie zawierał ani

krawędzi wielokrotnych, ani pętli (krawędzi, których obydwa końce są tym samym wierzchołkiem). Przypomnijmy, że kliką w grafie jest podzbiór wierzchołków tego grafu o tej własności, że każda para wierzchołków w tym podzbiorze jest połączona krawędzią. Kliką maksymalną jest taka klika, która nie jest całkowicie zawarta w innej klice.

Poniżej przedstawiamy algorytm Brona i Kerbosha, za pomocą którego możemy wygenerować wszystkie maksymalne kliki. Jest to algorytm rekurencyjny. Zakłada się, że pierwsze jego wywołanie następuje z parametrami: R, X — zbiory puste, P — zbiór wierzchołków badanego grafu. Korzysta się w nim z tablicy N zawierającej sąsiadów kolejnych wierzchołków.

```
#include <cstdlib>
#include <iostream>
#include <ctime>
#include <algorithm>
#include <vector>
#include <iterator>
using namespace std;
typedef vector<int> vertices; // sorted
typedef int vertex;
vector<vertices> N; // neighbors
void BronKerbosh(vertices R, vertices P, vertices X) {
  if (P.empty() and X.empty()) {
    for (vertex r : R) cout << r << " "; // report R as a maximal clique
    cout << endl;</pre>
  }
  else {
    vertices U, V;
    set_union(P.begin(), P.end(), X.begin(), X.end(), back_inserter(U));
    vertex u = U[rand() % U.size()];
    set_difference(P.begin(), P.end(), N[u].begin(), N[u].end(), inserter(V, V.begin()));
    for (vertex v : V) {
      vertices A, B, C, singleton = {v};
      set_union(R.begin(), R.end(), singleton.begin(), singleton.end(), back_inserter(A));
      set_intersection(P.begin(), P.end(), N[v].begin(), N[v].end(), back_inserter(B));
      set_intersection(X.begin(), X.end(), N[v].begin(), N[v].end(), back_inserter(C));
      BronKerbosh(A, B, C);
      remove(P.begin(), P.end(), v);
      auto it = find_if(X.begin(), X.end(), [v](const int &x){ return x > v; });
      X.insert(it, v);
    }
  }
}
int main(void) {
  srand(time(0));
  N.reserve(5);
  N[0] = \{1\};
  N[1] = \{0, 2, 3, 4\};
  N[2] = \{1, 3\};
  N[3] = \{1, 2, 4\};
 N[4] = \{1, 3\};
  BronKerbosh({}, {0, 1, 2, 3, 4}, {});
}
```

Przedstawiony program znajduje dla grafu:



wszystkie jego maksymalne kliki: {0, 1}, {1, 2, 3} oraz {1, 3, 4}.

Opracuj algorytm równoległy, a następnie zgodnie z nim zmodyfikuj powyższy program z wykorzystaniem biblioteki MPI. Dopisz także procedurę odczytującą graf z pliku tekstowego. Przyjmij następujący bardzo prosty sposób kodowania grafów za pomocą ciągu krawędzi poprzedzonego liczbą wierzchołków:

```
5

0 1

1 2

1 3

1 4

2 3

3 4
```

Ćwiczenie 4. Wyznacz co najmniej jedną wartość parametru c, dla której poniższa funkcja:

```
unsigned int hash(const char* s, unsigned int c) {
  unsigned int x = 0;
  for (unsigned int i = 0; s[i] != 0; ++i) {
    x <<= 3;
    x ^= s[i];
  }
  return (c/x) % 41;
}</pre>
```

będzie doskonałą funkcją haszującą (czyli przyporządkowującą każdemu elementowi liczbę całkowitą bez kolizji) dla zbioru słów kluczowych języka Python:

```
and
          del
                     from
                               not
                                          while
          elif
                     global
as
                               or
                                          with
                                          yield
assert
          else
                     if
                               pass
break
          except
                     import
                               print
class
                     in
                               raise
          exec
continue finally
                     is
                               return
          for
def
                     lambda
                               try
```

Ćwiczenie 5. Znajdź liczby rzeczywiste dodatnie x i y spełniające warunki: y-x < 2, 2x + 3y < 1 oraz $(0.2 - y)/(1.3 - x) - \sin(x)/x < -1.1008$.