Relatório - EP3 de MAC0323

Nome: Gabriel Haruo Hanai Takeuchi

NUSP: 13671636

Introdução

O problema consiste em encontrar o maior caminho em um dado digrafo não necessariamente acíclico.

Como compilar e executar

```
g++ -std=c++11 -Wall main.cpp -o a.out
./a.out < input.txt > output.txt
```

Montagem do grafo

O grafo foi montado a partir de um arquivo de entrada, que contém os fragmentos. A primeira linha contém a sequência original. A segunda linha contém o número de vértices n e o número k (*k-mer length*). As próximas n linhas contêm os fragmentos.

Para cada fragmento, foi criado um vértice no grafo. Para cada par de vértices u e v, a função `addArc(u,v)` verifica se **pelo menos** as k ou mais últimas letras de u são iguais às k ou mais primeiras letras de v. Se sim, uma aresta de u para v é criada e é adiciona na lista de adjacência de u.

Quebrando ciclos

Montado o grafo, é necessário quebrar todos os ciclos. Isso pode ser feito identificando e quebrando todas as arestas de retorno. Para isso, foi utilizada uma variação do dfs - `breakCycles()` e `breakCyclesHelper()`. A complexidade do algoritmo é $\mathcal{O}(V+A)$.

Identificando as fontes

Após a quebra dos ciclos, podemos adotar uma ordenação topológica do grafo. Foi utilizada a função `getFonts()`, que contabiliza a quantidade de arcos de entrada de cada vértice. Assim, temos os pontos de referência para descobrir os maiores caminhos do grafo. A complexidade do algoritmo é $\mathcal{O} (V+A)$.

Encontrando o maior caminho

A solução encontrada para o problema foi achar o maior caminho no grafo a partir de cada fonte e adotar o maior entre os maiores. Uma variação do algoritmo bfs foi utilizado para encontrar o maior caminho. A ideia é atualizar um vetor de distâncias `dist` sempre que um caminho maior para cada vértice for identificado. Isto acompanhado de um vetor `prev`, para resgatar o caminho.

Comparando a acurácia

O método de comparação foi feito a partir da função

```
double accuracyPercentage(const string& original, const string& longestPath)
{
  int originalSize = original.length();
  int longestPathSize = longestPath.length();
  int cont = 0; // counts the number of wrong characters
  int i = 0, j = 0;
  while (i<originalSize && j<longestPathSize) {
    if (original[i] != longestPath[j]) {
      cont++; j++;
    }
    else {
      i++; j++;
    }
  }
  return (1 - (double)cont / (double)originalSize) * 100;
}
```

O índice `i` percorre a sequência original e o índice `j` percorre a sequência encontrada.

`i` avança apenas quando o caracter atual de `i` é igual ao caracter atual de `j`.

`j` avança em ambos os casos.

`cont` conta o número de caracteres diferentes entre as duas sequências.

Testes

Foram feitas rodadas de testes com um grafo pequeno e um médio.

Grafo pequeno

Aqui, temos um grafo cíclico com 3 nós.

Input:

```
ACTCGTATACG
3 2
ACTCGT
CGTATA
TATACG
```

Output:

```
0: ACTCGT
Arcs:
(CGTATA , 3)
```

```
1: CGTATA
Arcs:
(TATACG , 4)

2: TATACG
Arcs:
(CGTATA , 2)

---------------------

After breaking cycles:

0: ACTCGT
Arcs:
(CGTATA , 3)

1: CGTATA
Arcs:
(TATACG , 4)

2: TATACG
Arcs:

---------------------

Original:      ACTCGTATACG
Longest path:  ACTCGTATACG
Accuracy:      100%
```

Para o grafo pequeno, vemos que o ciclo foi quebrado e a sequência original foi recuperada com total acurácia.

Grafo médio

Aqui, temos um grafo com 12 nós (o grafo do enunciado do EP):

Input:

```
ACTCGTAAATACATAACGATAC
12 2
ACTCGT
ATACATAA
TAACGA
ACGAT
TCGTA
AAATA
ATAAC
CGAT
GTAAATA
ACATAA
```

```
GATAC
GATAC
```

Output:

```
0: ACTCGT
Arcs:
(TCGTA , 4)
(GTAAATA , 2)

1: ATACATAA
Arcs:
(TAACGA , 3)
(AAATA , 2)
(ATAAC , 4)
(ACATAA , 6)

2: TAACGA
Arcs:
(ACGAT , 4)
(CGAT , 3)
(GATAC , 2)
(GATAC , 2)

3: ACGAT
Arcs:
(ATACATAA , 2)
(ATAAC , 2)
(CGAT , 4)
(GATAC , 3)
(GATAC , 3)

4: TCGTA
Arcs:
(TAACGA , 2)
(GTAAATA , 3)

5: AAATA
Arcs:
(ATACATAA , 3)
(TAACGA , 2)
(ATAAC , 3)

6: ATAAC
Arcs:
(ACTCGT , 2)
(TAACGA , 4)
(ACGAT , 2)
(ACATAA , 2)

7: CGAT
Arcs:
```

```
(ATACATAA , 2)
(ATAAC , 2)
(GATAC , 3)
(GATAC , 3)

8: GTAAATA
Arcs:
(ATACATAA , 3)
(TAACGA , 2)
(AAATA , 5)
(ATAAC , 3)

9: ACATAA
Arcs:
(TAACGA , 3)
(AAATA , 2)
(ATAAC , 4)

10: GATAC
Arcs:
(ACTCGT , 2)
(ATACATAA , 4)
(ACGAT , 2)
(ACATAA , 2)
(GATAC , 5)

11: GATAC
Arcs:
(ACTCGT , 2)
(ATACATAA , 4)
(ACGAT , 2)
(ACATAA , 2)
(GATAC , 5)

----------------------

After breaking cycles:

0: ACTCGT
Arcs:
(TCGTA , 4)
(GTAAATA , 2)

1: ATACATAA
Arcs:
(AAATA , 2)
(ATAAC , 4)
(ACATAA , 6)

2: TAACGA
Arcs:
(ACGAT , 4)
(CGAT , 3)
(GATAC , 2)
```

```
(GATAC , 2)

3: ACGAT
Arcs:
(ATACATAA , 2)
(ATAAC , 2)
(CGAT , 4)
(GATAC , 3)
(GATAC , 3)

4: TCGTA
Arcs:
(TAACGA , 2)
(GTAAATA , 3)

5: AAATA
Arcs:
(ATAAC , 3)

6: ATAAC
Arcs:
(ACATAA , 2)

7: CGAT
Arcs:
(ATACATAA , 2)
(ATAAC , 2)
(GATAC , 3)
(GATAC , 3)

8: GTAAATA
Arcs:
(ATACATAA , 3)
(TAACGA , 2)
(AAATA , 5)
(ATAAC , 3)

9: ACATAA
Arcs:

10: GATAC
Arcs:
(ATACATAA , 4)
(ACATAA , 2)
(GATAC , 5)

11: GATAC
Arcs:
(ATACATAA , 4)
(ACATAA , 2)

----------------------

Original:      ACTCGTAAATACATAACGATAC
```

```
Longest path:  ACTCGTAAATAACGATACATAAATAACATAA
Accuracy:      36.3636%
```

Aqui, vemos que a sequência foi similar até a 11ª posição. A acurácia infelizmente foi baixa.