

# **컴퓨터알고리즘과실습 프로젝트 보고서**

**컴퓨터알고리즘과 실습**  
**주종화 교수님**

2018112051 김도엽

2018112053 황종익

2018112061 남동호

2018112069 김준섭

## 1. 개요 및 데이터 생성

### 1-(1) 전제

**Given M number of short reads of length L, reconstruct the original sequence of length N that those shorts reads come from.**

\*N(Original sequence의 길이) : 2만개

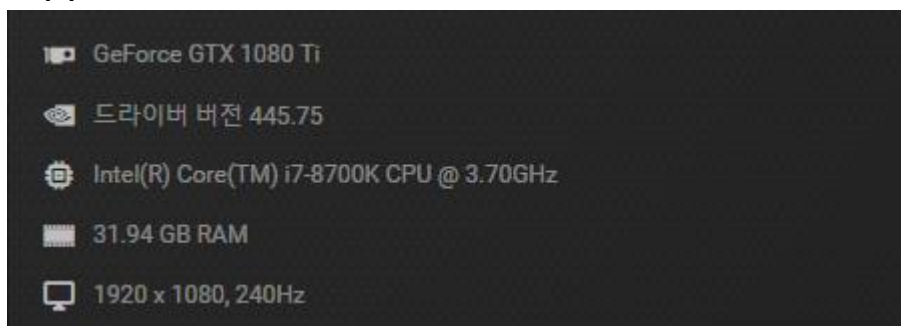
\*M(short reads의 개수) : 2000개

\*L(short reads의 길이) : 100

\*D(하나의 read에서 발생하는 mismatch의 개수) : 12개 이하

데이터 생성은, 아호 코라식을 사용하여 rand 알고리즘을 통해 생성되는substring의 반복 주기를 최대한 늘려 주기성을 최대한 제거해 주었다.

### 1-(2) 프로그램 실행 환경



부가적으로 windows 10을 설치한 환경에서 진행되었다.

## 2. 알고리즘 설명

### 2-(1) 김도엽 : De-Bruijn Graph(optimizer with Keyword Tree) + Graph

#### Traversal

#### Introduction

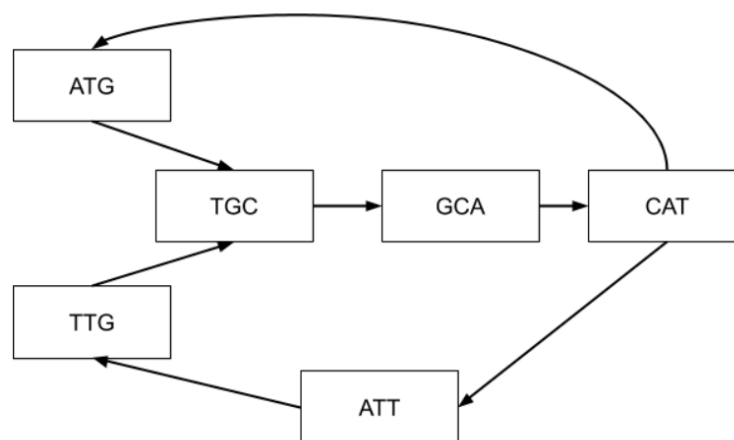
우선 기본적으로 De-Bruijn Graph를 활용하였다. 여기서 이 그래프를 탐색할 때 De-Novo의 방식을 차용하여 구현해주었다.

#### Generate De-Bruijn Graph

먼저 De-Bruijn Graph를 생성한다. 먼저 모든 short-read를  $K=27$ 인 K-mer로 재구성해주었다. 먼저 정점 집합을 구성해보자. De-Bruijn Graph의 경우 이 K-mer(길이가 3인 모든 substring 단 string은 short-read를 의미)들을 Node로 하는 그래프가 된다. 예를 들어 short-read의 집합이 {"ATGC", "ATTG", "GCAT"}로 존재하고  $K=3$ 인 K-mer로 구성을 한다면  $V=\{"ATG", "TGC", "ATT", "TTG", "GCA", "CAT"\}$ 의 형태가 되는 것이다. 이때 집합  $V$ 가 정점 집합이 되는 것이다.

이제 간선 집합을 구성해보자. 이 그래프에서 간선은 방향 간선을 띤다. (즉 방향그래프이다.) 간선의 경우 string  $S, T$ 가 있으면  $S$ 에서 길이가  $(K-1)$ 인 suffix,  $T$ 에서 길이가  $(K-1)$ 인 prefix가 같다면  $S \rightarrow T$ 꼴의 간선이 존재한다. 즉 "TGC", "GCA"가 존재하면 이를 그림으로 표현하면 길이  $(K-1)$ 의 suffix, prefix가 "GC"로 서로 같으므로 "TGC"  $\rightarrow$  "GCA"꼴의 간선이 존재한다.

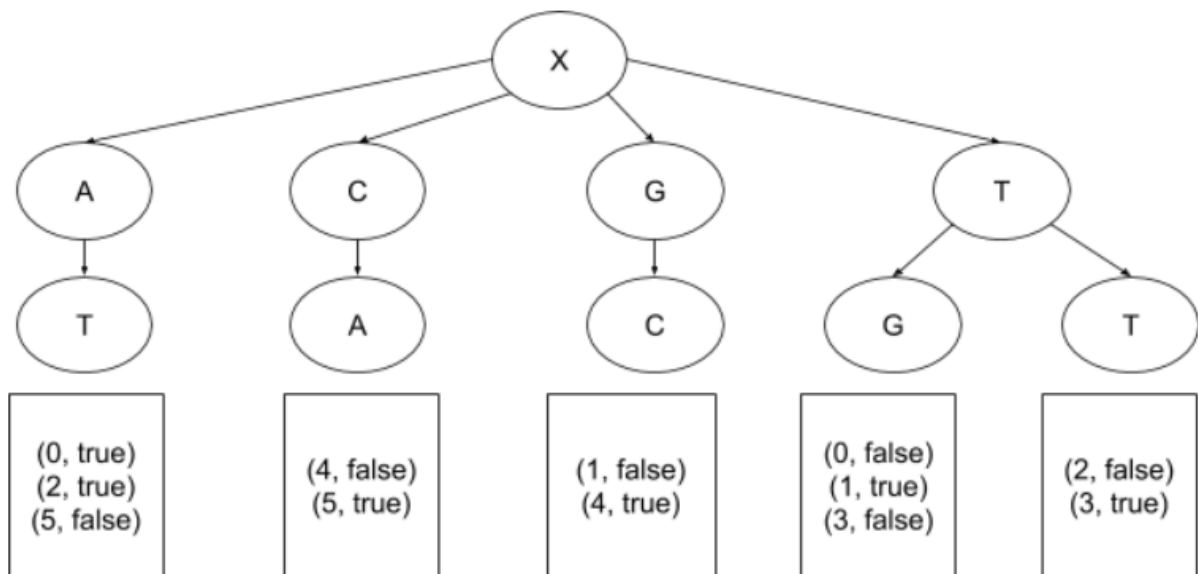
위 같은 과정을 통해 아래 <De-Bruijn Graph>의 그래프 형태가 만들어진다.



<De-Bruijn Graph>

#### Optimize

이제 이 그래프를 구현을 할 때 간선 생성의 최적화를 위해 Keyword Tree을 사용한다. 위에서의 집합  $V$ 의 내용들을 2중 for로 돌게 되면 시간 복잡도가  $O(|V|^2 K)$ 가 된다. 하지만 우리가 할 일은 suffix와 prefix가 같은 그룹(즉, "AGC", "GCC", "TGC"가 하나의 그룹이 됨)만 보면 되는데 만약 이 그룹이 균등하게  $\alpha$ 개의 그룹으로 나뉘었다면 휴리스틱하게  $O\left(\left(\frac{|V|}{\alpha}\right)^2 \times \alpha\right) = O\left(\frac{|V|^2}{\alpha} \cdot K\right)$ 의 시간 복잡도를 가질 수 있다. 하나의 suffix, prefix를 모두 Keyword Tree에 삽입하고 리프 노드(삽입한 모든 단어의 길이가 같기에 리프만 확인)에는 각 고유 번호와 이 문자열이 suffix인지, prefix인지를 저장해주었다. (코드에서 인덱스가 고유 번호, true: prefix, false: suffix) 예시에서의  $V$ 집합을 Keyword Tree에 삽입한 결과는 아래와 같다. (a, b, c일 때  $a \rightarrow 0$ ,  $b \rightarrow 1$ ,  $c \rightarrow 2$ 의 인덱스를 가지게 됨 즉 왼쪽부터 0, 1, 2, 3...를 부여)



이제 이 정보를 가지고 리프에 있는 노드들끼리 서로 다른 (하나의 prefix이고, 나머지 하나는 suffix인) 형태를 가지면 간선을 위에 설명한 것처럼 연결해준다.

또한 최적화를 위해서 K-mer를 생성하는 과정에서 V에 들어있는 요소들의 중복들을 모두 제거해주었다.

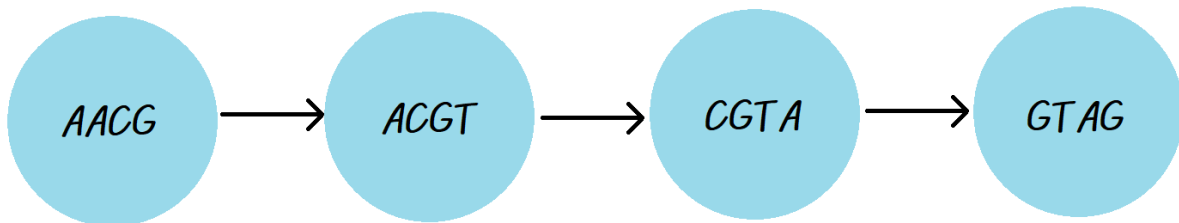
## Graph Traversal

위에서 최적화를 진행하면서 정점들의 중복을 제거해주었는데 이때 그 개수를 따로 저장해 두었다. 이를 바탕으로 한 정점을 방문할 수 있는 최대 횟수를 지정해 두었고 이 최대 횟수를 활용하여 모든 간선을 최대한 다 지나가게끔 구현하였다.

최종적인 결과물은 De-Novo 의 방식으로 표본이 적으면 합칠 수 없는 문제점 때문에 이 경로들의 집합으로 결과를 내었고 그중 N50을 택해 알고리즘의 정확성을 평가했다.

## 2-(2) 황종익 : Bloom Filter와 De Bruijn Graph를 이용한 알고리즘

기본적으로 De novo 방법이며, Bloom Filter라는 것을 이용해 k-mer들이 이어지는 De Bruijn Graph에서 확실하게 이어지는 것들이 아닌 것을 없애 주는 알고리즘이다.



De Bruijn 그래프란, 위 그림처럼 Vertex들이 길이-1만큼 겹치는 것들을 방향 그래프로 이어 엮기 서열들이 이어질 가능성이 있는 Edge들을 추가해주는 그래프이다. 이러한 De Bruijn 그래프의 edge들을 Bloom Filter를 이용해 줄여 나가는 방식으로 진행된다.

알고리즘 실행 과정과 함께 원리를 설명해 보게 되면 다음과 같다.

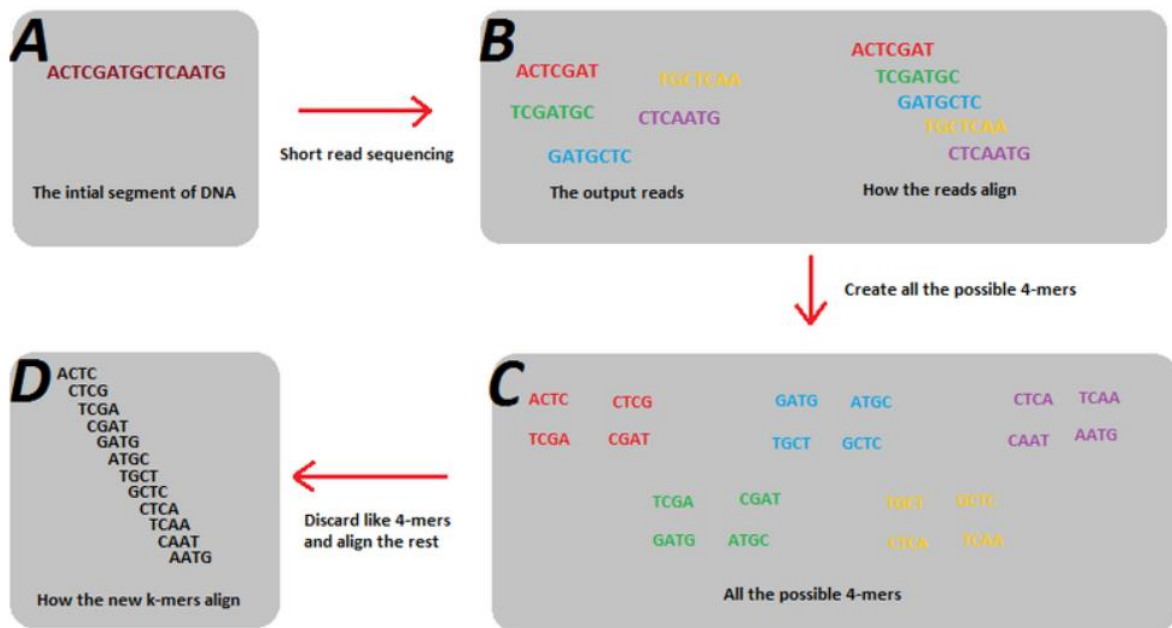
1) 입력받은 shortRead들을 k길이의 k-mers들로 만들어낸다.

예)

Read: AGATCGAGTG

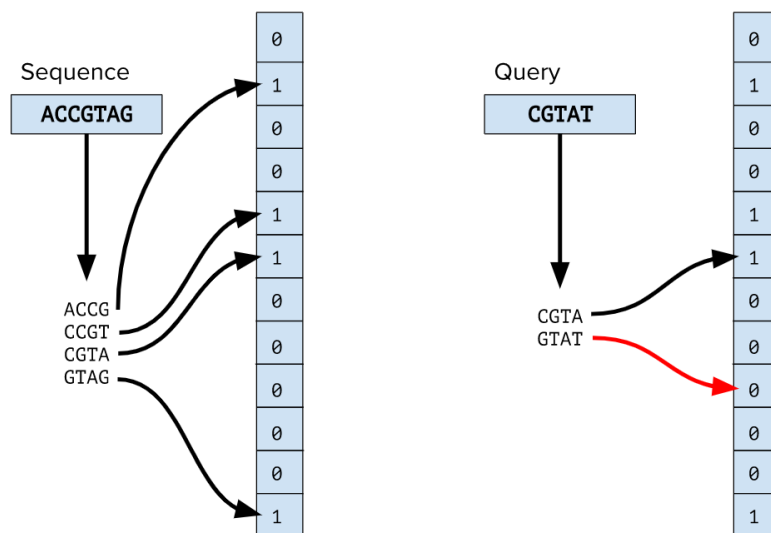
3-mers: AGA GAT ATC TCG CGA GAG AGT GTG

이 때, k의 값은 shortRead길이보다 크지 않으며, k값을 크게 하면 중복되는 k-mer들이 줄어들게 되지만, k-mer 자체의 개수가 적어져 그래프 생성에 어려움을 겪을 수 있다는 단점이 있다. k값이 작은 k-mer의 경우 이전 경우의 반대가 된다.



k-mer의 생성 필요성에 대해서는, De Bruijn 그래프의 특성이 작용하게 된다. 위 그림 B와 같은 shortRead에 대해 De Bruijn 그래프를 생성하게 되면 처음의 DNA 시퀀스로 복구를 할 수 없다. 하지만 과정 C와 같이 shortRead를 4-mer로 쪼개게 되면 De Bruijn 그래프를 통해 DNA 시퀀스를 복구해낼 수 있다.

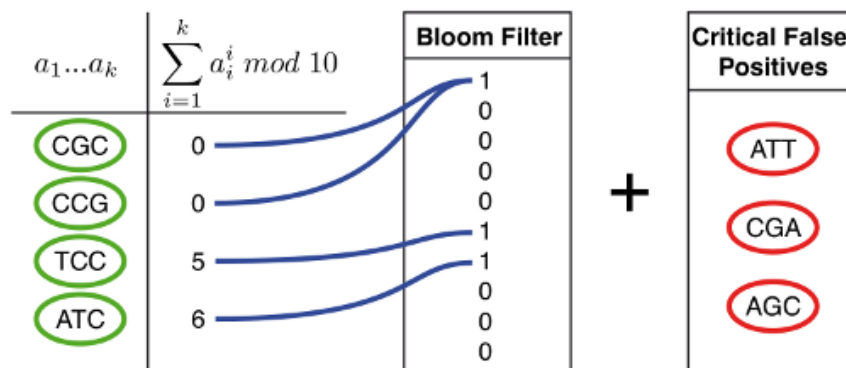
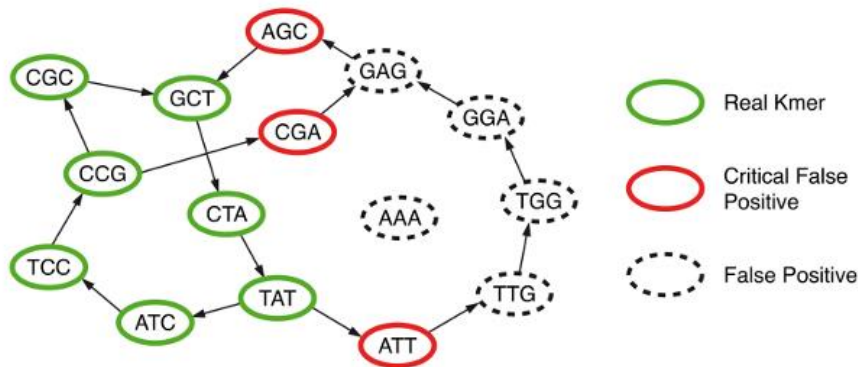
2) k-mers에 대해 Bloom Filter를 적용한다.



Bloom Filter는 입력받은 k-mers들을 `vector<bool>` 형태로 이루어진 1비트 해시 테이블 배열에 두 가지 해싱을 적용하여 저장하게 된다. 다른 map, set과 각각의 key나 value값을 가지고 있게 되는데, 이러한 방식을 사용하게 되면 하나의 1비트 배열에 모두 저장이 가능하기 때문에 메모리 효율, 공간적으로 많은 이득을

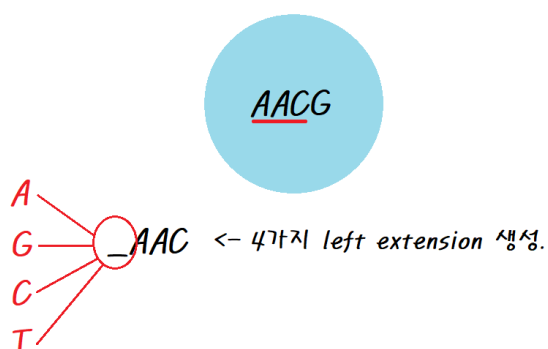
볼 수 있다.

이런 과정을 거치게 되면, 쿼리로 들어오는 문자열에 대해 해시값을 만들어 해당 위치에 값을 비교해 그것이 확실히 존재하지 않는 k-mer인지에 대한 구분을 가능하게 해 확실히 배제(False Positive)할 수 있기 때문에 De Bruijn 그래프를 더욱 단순화시킬 수 있게 된다.



위 그림과 같이 Criticcal False Positives들을 탐색에서 제외할 수 있다. 길이 끊어진 곳 또한 false positives가 되면서 그래프 탐색에 필요한 edge들이 줄어들게 된다.

3) BFS를 통해 contigs들을 생성해낸다.



탐색을 시작하는 정점들을 찾는 방법은, 아래 그림과 같은 경우에, 4가지 문자를 왼쪽에 이은 left extension들을 생성하고, bloom filter에 넣어 확인해보아서 존재하지 않는 경우가 될 때, 시작 정점이 된다.

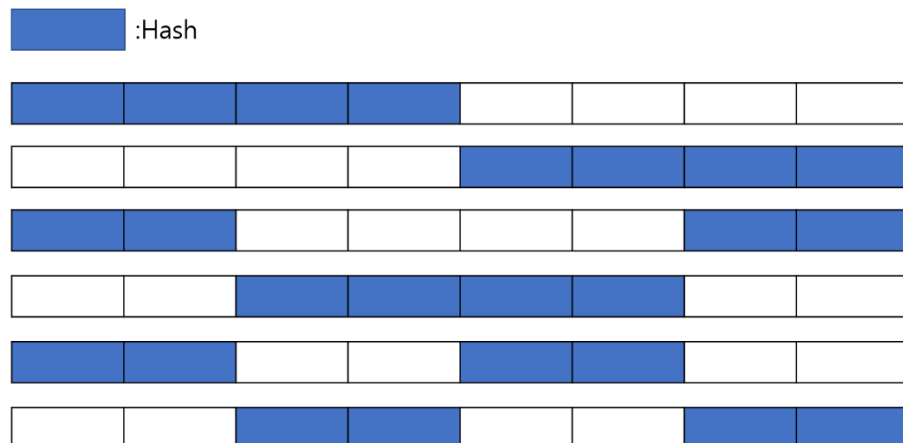
이 시작 정점을 필두로 BFS를 돌리며 right extension들을 생성해 나가며 존재하는 kmer들을 이어주게 되면 결국 긴 길이를 가지는 contig들이 생성되게 된다.



### 2-(3) 남동호 : MAQ 알고리즘을 응용한 Hash 기반 알고리즘

일반적으로 해쉬 기법을 통한 스트링 매칭에서는 해쉬 값의 충돌이 일어나는 구간에 대하여 스트링 원소들의 일대일 비교를 통해 스트링을 탐색한다. 하지만 Reference DNA를 통한 DNA 복원에서는 단일 염기 다형성(SNP)라는 문제점이 발생한다. 해쉬 함수를 통해 스트링의 해쉬값을 추출하는 방법에서는 스트링의 원소에 변이가 발생하면 기존의 해쉬값과 완전히 동떨어진 해쉬값이 추출될 수 있다는 것이다. 이러한 문제점으로 단순히 해쉬 함수를 적용시키는 방법으로는 일정 수준의 MissMatch를 허용하는 방식으로 DNA를 복원할 수 없다. 따라서 해쉬 기법을 사용한 DNA 복원에서 가장 중요한 해결 과제는 '어떠한 방식으로 일정 수준의 변이를 허용하는가' 이다.

MAQ 알고리즘은 해쉬를 통한 DNA복원에 사용되는 알고리즘이다. 이 알고리즘의 기본적인 내용은 ShortRead 서열로 해시테이블을 생성하고 Reference DNA를 검색하는 것이다. 이 알고리즘의 핵심은 해시 테이블 구성에 있다. 만약 ShortRead 서열 전체를 Key로 해시 테이블을 구성한다면 MissMatch를 허용할 수 없으므로 ShortRead를 여러 부분으로 나누어 해시 테이블을 생성한다. 이때 단순히 임의의 길이, 방식으로 ShortRead를 분할하는 것이 아닌, MissMatch 개수를 고려한 분할 방식을 선택해야 한다. 다음 이미지는 ShortRead의 길이가 8이고 MissMatch의 최대 허용 개수가 2개일 경우, 해시 테이블을 구성하는 방식이다.



위 그림을 보면 하나의 ShortRead에 대해 6개의 Key를 갖는 해시 테이블을 생성한다는 것을 알 수 있다. 또한 위 그림에서 가장 중요한 것은 6개의 분할 스트링 중 최소한 2개가 일치할 경우 MissMatch가 2개 이하라는 것이 증명된다는 것이다. 이는 임의의 2개의 분할 스트링에 대해 모두 성립하는 특성이다. 즉 MAQ 알고리즘에서 가장 중요한 것은 스트링의 분할에 있어서 임의의 N개의 해쉬값이 충돌한다면 MissMatch가 특정 범위 안에 존재할 수 있도록 분할 구간을 설정하

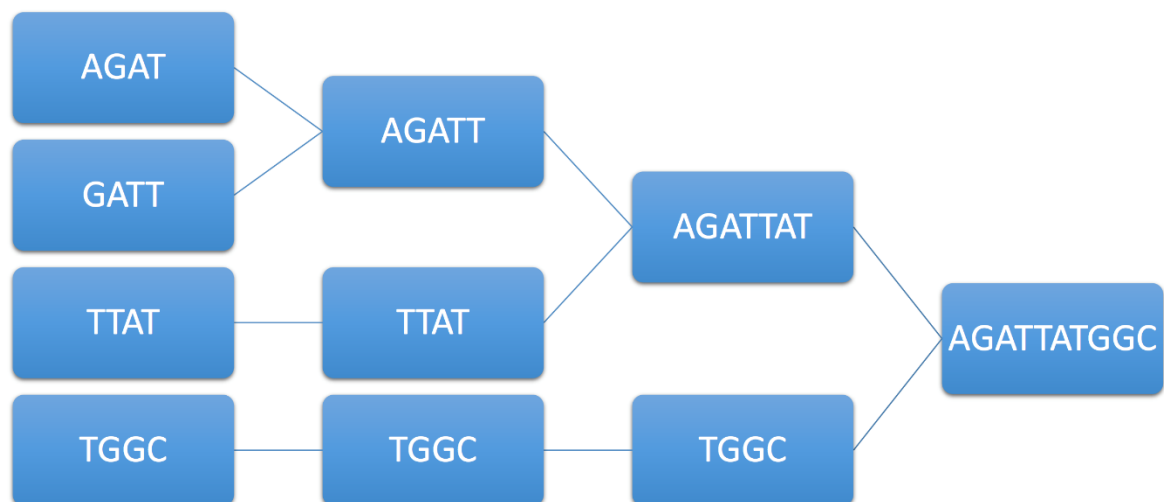
는 것이다.

## 2-(4) 김준섭 : Greedy 와 De-Novo

Short read 간의 overlap이 클수록 같은 DNA의 부분일 것이라는 생각으로 Greedy 알고리즘을 이용하여 de novo sequencing을 구현하였다. 우선 모든 short read를 벡터에 넣는다. 그리고 벡터의 모든 원소들 간의 문자열 비교를 통해 overlap을 구한다. 가장 overlap이 큰 두 short read를 고른 후 merge한다. 이처럼 문자열 비교를 통해 overlap을 구하고, 두 short read를 merge하는 과정을 벡터의 short read가 하나 남을 때까지 반복한다. 마지막으로 남은 원소가 복구한 DNA가 된다.

short read의 길이를  $k$ , short read의 개수를  $n$ 이라고 할 때, 두 short read 간의 overlap을 구하는 것은 최악의 경우 문자열을 전부 iterate해야 하기에  $O(k^2)$ 이다. 그리고 merge를 위한 overlap의 최댓값을 구하기 위해 모든 short read 간의 비교를 해야하므로  $O(n^2)$ 이 걸린다. 이 과정을 한 번 할 때 마다 merge가 한번 이루어지므로, 총  $n$ 번 반복된다. 따라서 최종 수행시간은  $O(k^2n^3)$ 이다.

Greedy 알고리즘의 효율을 증가 시키기 위해 따로 두 가지 기법을 사용하였다. 우선 모든 문자열이 아예 똑같은 short read가 존재할 경우 서로 같은 파트라고 인식하여 하나만 남기고 제거하였다. 이를 수행할 경우 short read의 개수인  $n$ 을 줄일 수 있어 수행시간에 큰 이득을 볼 수 있다. 그리고 Greedy 알고리즘 특성상 잘못된 결과를 얻을 확률이 높기 때문에 threshold를 사용하였다. 두 short read의 overlap의 크기가 설정한 threshold 보다 작을 경우 merge가 이루어지지 않게 하여 overlap의 크기가 일정 이상 큰 short read들부터 merge가 이루어지도록 하였다. 단, 이 경우 sequencing 과정이 아예 정지할 수 있기 때문에, 그 전 단계에 비하여 merge가 이루어지지 않았더라면 threshold의 값을 차감하였다.



-Greedy 알고리즘을 통해 올바르게 DNA sequencing이 이루어진 경우, overlap의 크기가 큰 short read들부터 merge가 이루어져 최종 DNA를 얻었다.

### 3. 결과 및 성능

#### 3-(1) 김도엽 : De-Bruijn Graph + Graph Traversal

사실 이 알고리즘의 정확도를 표기하기 너무 힘든데 위에처럼 휴리스틱하게 표기한다.

균일하게 분포되어 그룹수가  $\alpha$ 개 라면 그래프를 생성하는데  $O\left(\frac{|V|^2}{\alpha} \cdot K\right)$ 만큼 걸리고 그 뒤 그래프 탐색시 모든 간선을 모두 지나는 상황이 최악이므로  $O(E)$ , 이므로 아마도  $O\left(\frac{|V|^2}{\alpha} \cdot K + E\right)$ 정도 나올 것이다.

또한 De-Novo의 특성상 완벽한 결과물을 내기 힘들기 때문에 contig라는 원본 DNA의 부분 염기 서열을 출력하는 방식으로 구현하였다. 이렇게 되면 성능 평가가 힘들어 지는데 이를 해결하기 위해 N50이라는 개념을 도입하게 된다. 이 N50이라는 요소는 3-(2)에 자세히 설명되어있지만 간략하게 소개를 한다. N50은 De-Novo를 평가하는 요소 중 하나로 contig들을 길이순으로 내림차순 정렬하고 그 길이의 누적합이 원본 DNA의 절반 이상을 넘는 최초의 contig의 길이를 의미한다. 이 길이가 길면 길수록 해당 알고리즘의 효율이 높다고 본다고 한다. 단 하나만 존재하는 contig 또는 contig중 가장 긴 길이의 contig에 대해서 매칭 시키고 "길이 차"를 통해 계산을 진행하였다.

실험 1) N=20000, M=2000, L=100

```
Length of Reference : 20000
Number of Shortreads : 2000
Length of Shortread : 100

100    % complete

It takes 0.627seconds

Accuracy of this algorithm is 80.545%

End Process
Show "compare.txt" file!
```

일치율 80.545%정도인 것을 확인 할 수 있다.

실험 2) N=25000 (M, L 동일)

```
Length of Reference : 25000
Number of Shortreads : 2000
Length of Shortread : 100

100    % complete

It takes 1.135seconds

N50 is a measure to describe the quality of assembled genomes that are fragmented in contigs of different length.
This value is meaningful when it is more than 1/4 of the my original DNA length.
Matching Rate : 24.58%
N50 of this algorithm is 24992

End Process
Show "compare.txt" file!
```

원본 문자열의 길이가 늘어나 정확도가 떨어짐과 동시에 contig 들이 여러 개 생성됨을 볼 수 있다. 따라서 N50 을 통해 성능이 측정되었다.

### 실험 3) N=30000 (M, L 동일)

```
Length of Reference : 30000
Number of Shortreads : 2000
Length of Shortread : 100

100    % complete

It takes 0.532seconds

N50 is a measure to describe the quality of assembled genomes that are fragmented in contigs of different length.
This value is meaningful when it is more than 1/4 of the my original DNA length.
Matching Rate : 24.28%
N50 of this algorithm is 29951

End Process
Show "compare.txt" file!
```

30000 으로 값이 늘어나니 값이 원본으로 복원되는 모습이 상당히 감소했다. 결국 25000 처럼 N50 을 통해 측정하게 되었다.

### 결론



### 3-(2) 황종익 : Bloom Filter와 De Bruijn Graph를 이용한 알고리즘

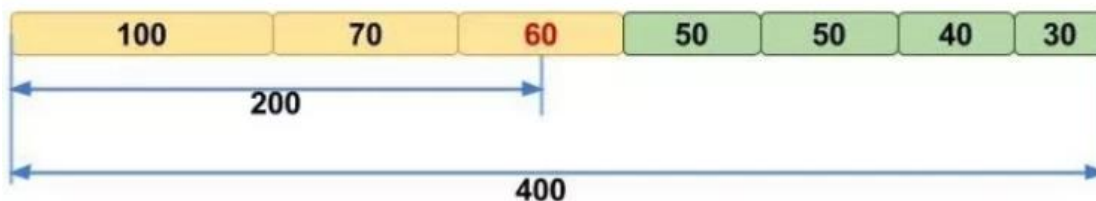
주어진 환경에서 BFS 기반, kmer들을 만들어내기 때문에  $V$ 는 가장 많아도  $n$ 개, 엣지는  $k$ 값을 kmer들이 서로 겹치지 않을 정도로 주었기 때문에 각 Vertex마다 2개씩, 각 노드들의 존재를 확인하는데에는 해싱을 사용하기 때문에  $O(\log(n))$ 이 걸린다고 계산했을 때, contig들을 생성하는 그래프 탐색 시간복잡도는 대략  $O(\log(n)) * O(|V|+|E|) = O(\log(n)) + O(n + (2n)) = O(n\log(n))$  이 된다.

**성능 평가 요소 : contig에 대한 N50값, myDNA와의 가장 긴 contig 일치율, 소모 시간**

N50은 de novo assembly의 품질을 정의할 때 사용되는 수치로, 주어진 assembly의 contig의 set들의 길이를 모두 합쳤을 때 절반 길이를 구하고 가장 긴 contig 서열부터 차례차례 합산된 누적값이 이들의 절반 길이에 해당하는 contig 길이를 의미한다.



1a. Contigs, sorted according to their lengths.



1b. Calculation of N50 using sorted contigs.

위 그림은 전체 contig set 길이가 400이고, 길이순으로 더해가다보면, 길이 60인 contig에서 400의 절반인 200이 넘어가게 되므로, N50수치는 60이 된다.

**1) 기존 전제 조건 (  $N=20000$ ,  $M=2000$ ,  $L=1000$ 일 때 )**

```

kmers의 수: 154000
filterSize : 1669493 numHashFunctions : 7
블룸 필터 생성중...
거짓 긍정들을 찾아내고 있습니다...
그래프 순회 중...
initKmersSize : 1
Starting kmer: AAGTTGCGAGATGAGCGTGCATCG, 진행도 : 1/1
소모 시간 : 0.679seconds
De Bruijn 그래프 크기(바이트) : 1669493

원본 myDNA의 길이 :20000
복구된 가장 긴 contig의 길이 :20000

다음에 나오는 아래의 정확도는 원본 길이의 1/4배정도가 복구되었을 때만 의미가 있는 값입니다.
그보다 짧다면 더 아래의 N50값을 확인해주세요.
DNA 일치율 : 100%
N50 : 20000
(유의미한 경우에 속한다.) DNA 일치율 : 100%

```

일치율 : 100%, N50값 : 20000, 소모 시간 : 0.679초

완벽히 myDNA를 복원해냈다. contig의 개수가 단 1개이다.

## 2) N만 25000으로 변경했을 때

```

kmers의 수: 154000
filterSize : 1669493 numHashFunctions : 7
블룸 필터 생성중...
거짓 긍정들을 찾아내고 있습니다...
그래프 순회 중...
initKmersSize : 4
Starting kmer: GTGCCACGGTGTAGACGTGTGTCG, 진행도 : 1/4
Starting kmer: TCGTGACAGCACAGCTGCAGCACT, 진행도 : 2/4
Starting kmer: GAGACACTGTATGCTCGCGAGAGA, 진행도 : 3/4
Starting kmer: GCTGTCACGCACTGATCAGCACGC, 진행도 : 4/4
소모 시간 : 0.724seconds
De Bruijn 그래프 크기(바이트) : 1669493

원본 myDNA의 길이 :25000
복구된 가장 긴 contig의 길이 :16614

다음에 나오는 아래의 정확도는 원본 길이의 1/4배정도가 복구되었을 때만 의미가 있는 값입니다.
그보다 짧다면 더 아래의 N50값을 확인해주세요.
DNA 일치율 : 66.456%
N50 : 16614
(유의미한 경우에 속한다.) DNA 일치율 : 66.456%

```

일치율 : 66.456%, N50값 : 16614, 소모 시간 : 0.724초

원본 길이가 늘어나니 k-mer들이 잘 겹치지 않아 정확도가 내려간 것을 볼 수 있다. contig의 개수는 4개이다.

소모 시간 : 0.724초

## 3) N만 30000으로 변경했을 때

```

kmers의 수: 154000
filterSize : 1669493 numHashFunctions : 7
블룸 필터 생성중...
거짓 긍정들을 찾아내고 있습니다...
그래프 순회 중...
initKmersSize : 10
Starting kmer: AGCAGCTGTAGATCTATCTCACAG, 진행도 : 1/10
Starting kmer: TCGATCGTGCAGCGTGGACTTACG, 진행도 : 2/10
Starting kmer: GTCAGTACAGATACATATCGACTA, 진행도 : 3/10
Starting kmer: GCGTGTGCGCACTGTCTCAGCGTGA, 진행도 : 4/10
Starting kmer: GCAGAGAGAGTCGCTCTGCGCATC, 진행도 : 5/10
Starting kmer: AGCGTACTAGCATATCGTATGCTC, 진행도 : 6/10
Starting kmer: TGATCTGTCTAGACGTCTATACTC, 진행도 : 7/10
Starting kmer: CGCACATGCAGTGTGTCAGCGCAG, 진행도 : 8/10
Starting kmer: GACTATACTGAGACACGCAGCTGT, 진행도 : 9/10
Starting kmer: GTCTGCACTGACGCTAGCTCATCG, 진행도 : 10/10
소모 시간 : 0.786seconds
De Bruijn 그래프 크기(바이트) : 1669493

```

원본 myDNA의 길이 : 30000  
복구된 가장 긴 contig의 길이 : 9209

다음에 나오는 아래의 정확도는 원본 길이의 1/4배정도가 복구되었을 때만 의미가 있는 값입니다.  
그보다 짧다면 더 아래의 N50값을 확인해주세요.

DNA 일치율 : 30.6967%

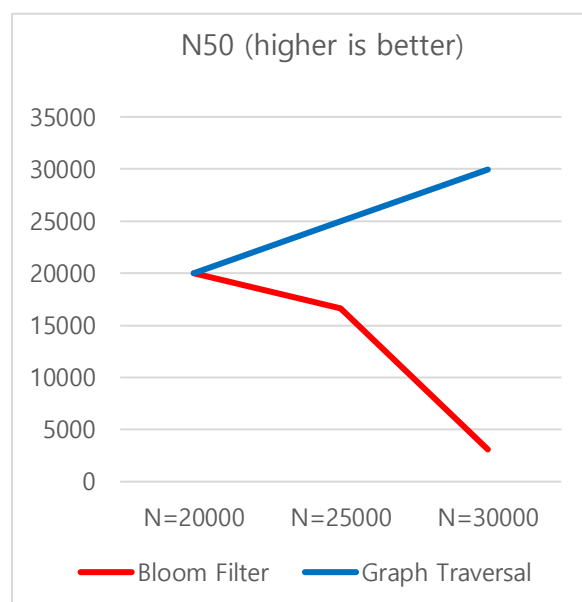
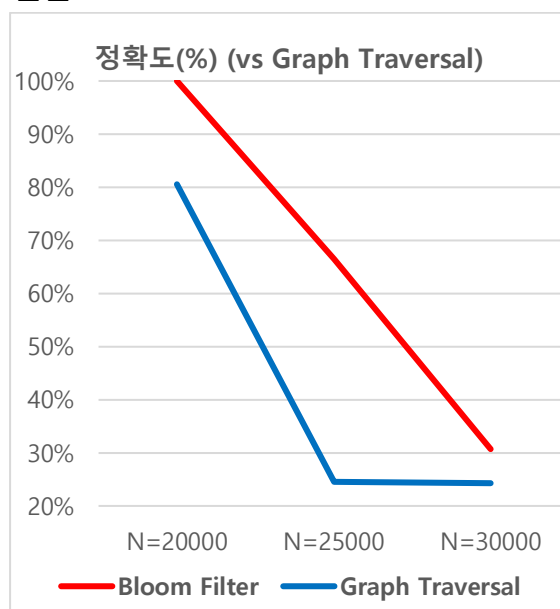
N50 : 3089

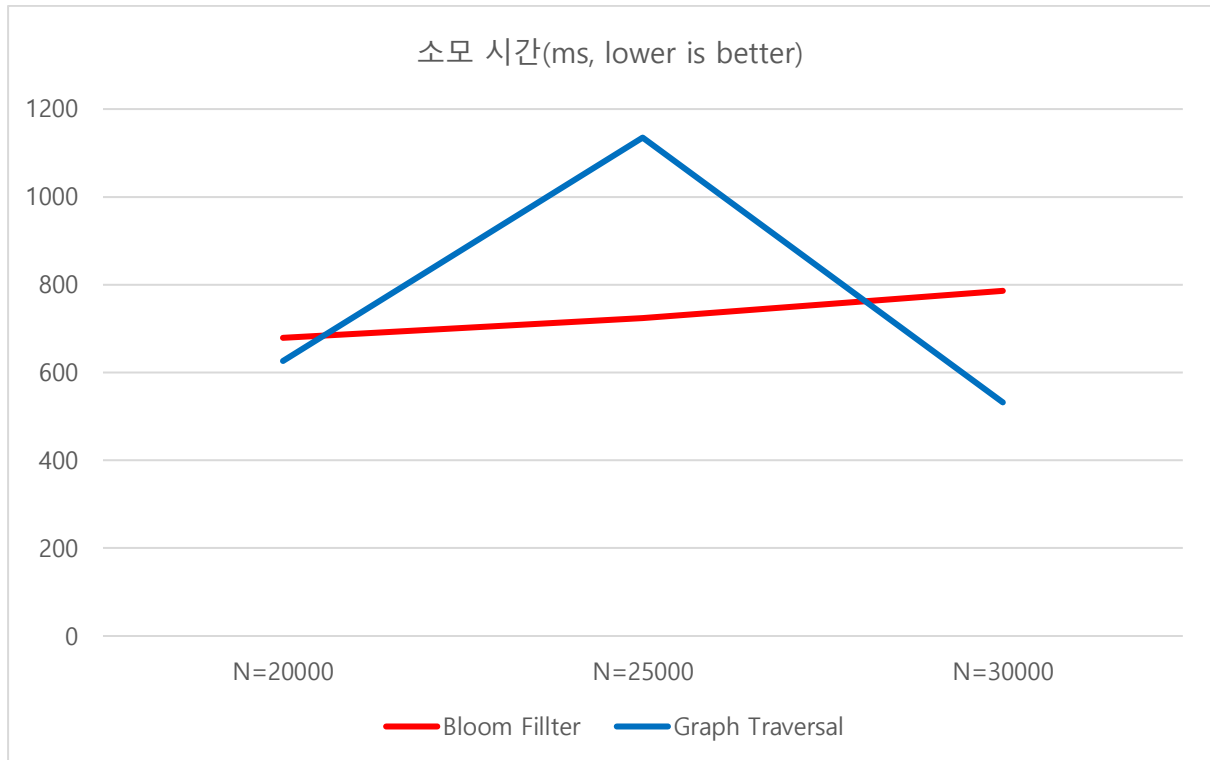
(유의미한 경우에 속한다.) DNA 일치율 : 30.6967%

일치율 : 30.6967%, N50값 : 3089, 소모 시간 : 0.786초

가장 긴 contig의 길이가 myDNA의 길이의 절반을 넘지 않는 정도가 되기 때문에 N50값이 9209가 아닌 다른 값이 되었다. contig의 개수는 10개이다.

## 결론 :





myDNA의 길이가 길어질수록 소모 시간 또한 증가하였으며, 정확도는 낮아지고, N50의 수치 또한 낮아졌다. 정확도와 N50 수치는 원본 DNA길이 대비 shortRead의 개수와 길이의 곱이 커질수록 더욱 정확하고, 큰 값을 가지게 되었다.

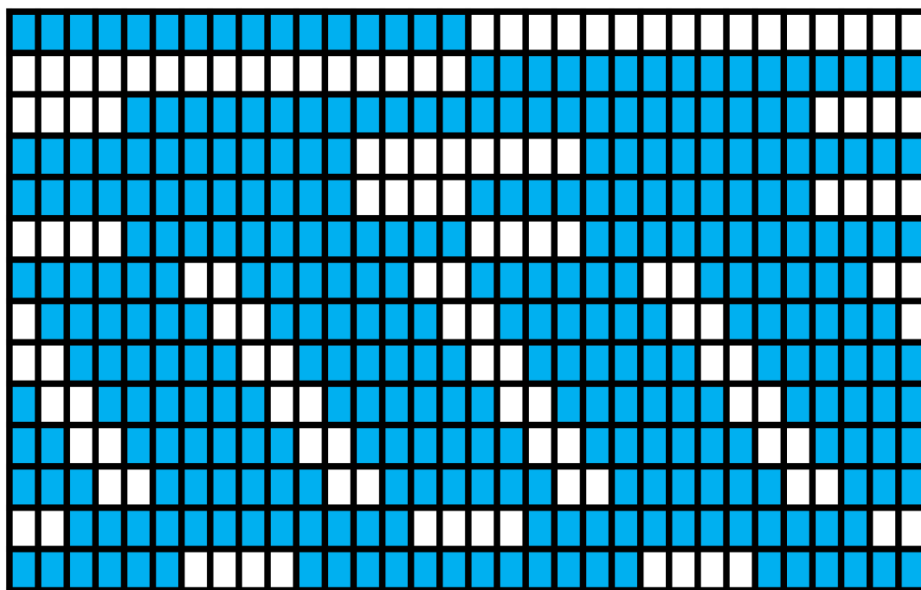
아쉬운 점으로는, 확실하게 이어지는 부분이 아니면 원본 DNA의 길이만큼으로 복구할 수 없어 아쉬웠고, 조금 더 유연하게 kmer들을 이어 나갈 수 있었다면 더 긴 contig들을 생산해낼 수 있지 않았을까 라는 생각을 하게 되었다.



### 3-(3) 남동호 :

ShortRead 분할 해싱을 이용한 DNA 복원은 각각의 ShortRead에 대해 해시 테이블을 생성하고 Reference 서열을 검색하는 방식이다. 이 과정에서 해시 테이블 생성, 해시 테이블 검색, 스트링 분할이 이루어지지만 C++의 unordered\_map은 해시 테이블의 생성, 삽입, 삭제의 실행이  $O(1)$  시간 동안 이루어지고, 스트링 분할 또한 분할 개수가 고정되어 있으므로 분할 시간은  $O(1)$ 이다. 게다가 분할 개수가 고정됨에 따라 Reference DNA와 ShortRead간의 비교 횟수 또한 상수이다. 따라서 ShortRead의 길이가  $l$ , Reference DNA의 길이가  $L$ , ShortRead의 개수가  $M$ 일 때 알고리즘의 최악 시간 복잡도는  $O(L \times M)$ 이다. 단순히 시간 복잡도를 비교했을 경우 Trivial의  $O(L \times M \times l)$ 보다 빠르다고 할 수 있지만 실제 구현 과정에서 스트링의 분할과 해시테이블 생성, 검색에 소요되는 시간이 크기 때문에 신속한 복원에 어려움이 존재한다. 또한 분할 개수가 고정되어 있지만 ShortRead와 MissMatch의 최대 허용수가 증가하면 분할의 개수 또한 큰 폭으로 증가(관계식을 구할 수 없어서 이렇게 표현하였습니다.)하기 때문에 실질적으로 Trivial보다 긴 시간이 소요된다. MAQ의 특성상 실질적으로 모든 MissMatch의 분포를 커버할 수 없기 때문에 정확성 측면에서도 한계가 존재한다고 볼 수 있다.

팀 프로젝트의 공통 사항은 Reference의 길이=20000, ShortRead의 길이=100 (100은 회의 후 수정된 값. 기존 값은 32. 실제 구현은 32를 기준으로 함), ShortRead의 개수=2000, 허용하는 MissMatch 개수=4이다. 위에서 설명한 MAQ 알고리즘의 특성을 고려하여 ShortRead의 길이가 32인 경우에 대해 분할 구간을 다음과 같이 설정하였다.

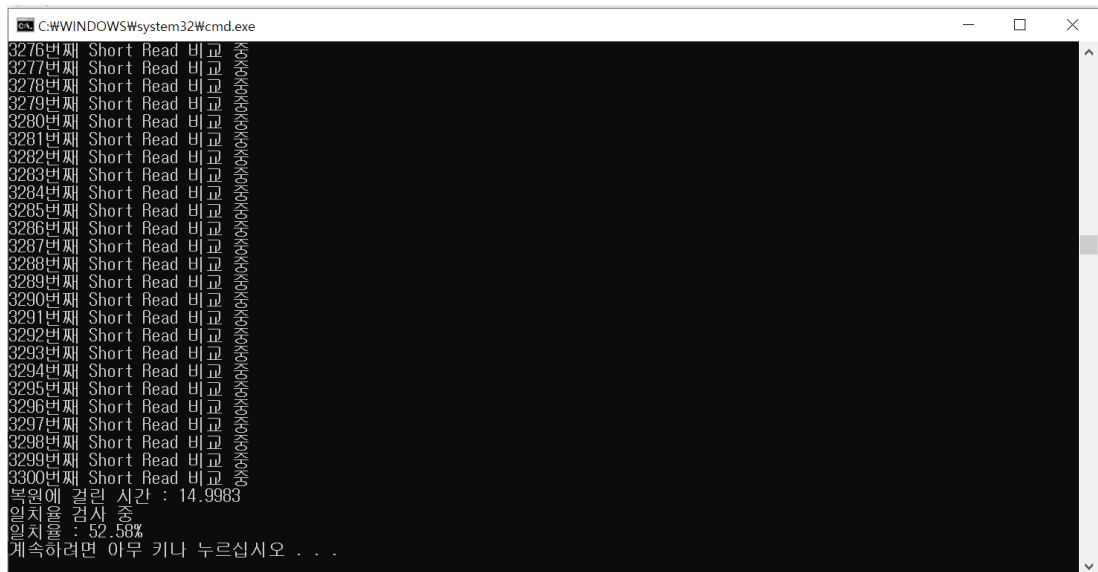


위와 같은 분할을 통해 최소한 2개 이상의 해시값 충돌이 발생한다면 MissMatch의 개수가 4개 이하라는 것이 증명된다. 다만 실행 시간을 줄이기 위해 분할의 개수를 조절하는 과정에서 모든 MissMatch의 분포에 대해서 대응하지 못하는 한계점이 존재한다. 위와 같은 분할 방식을 통해 구현한 DNA 복원 과정은 다음과 같다. 1) 길이가 32인 ShortRead를 분할하여 해시 테이블을 생성 2) 전체 Reference DNA에 대하여 32단위로 위와 같이 분할하여 해시 테이블을 검색 3) 해시값 충돌이 2회 발생하면 해당 위치에 ShortRead 삽입

예시 이미지로는 Reference DNA 길이=100000, ShortRead 길이=32, ShortRead 개수=3300개,

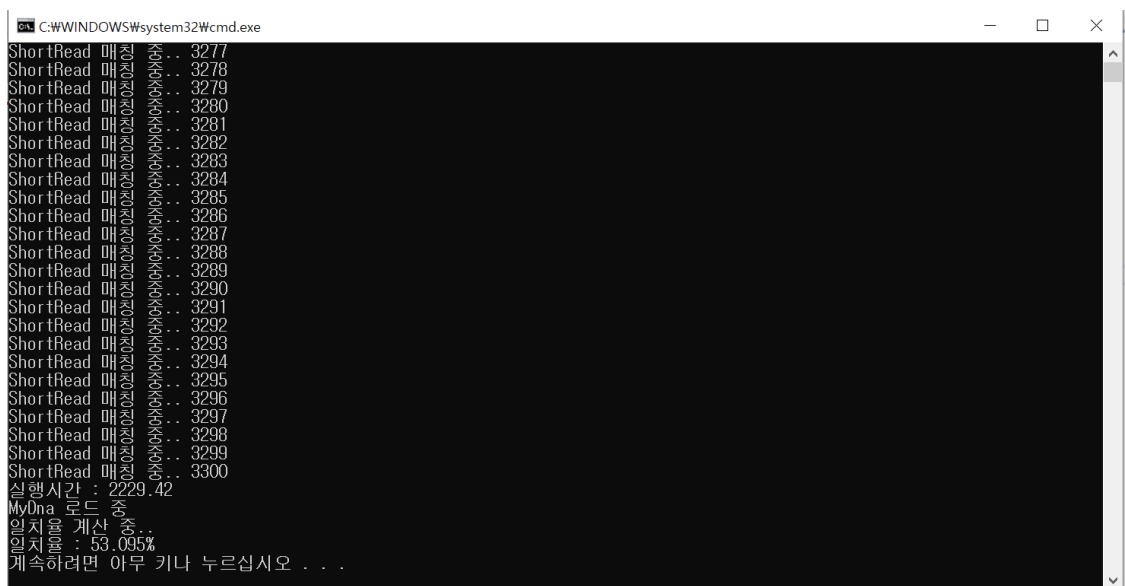
최대 허용 MissMatch개수=12개 이하. 이와 같은 조건으로 MAQ알고리즘과 Trivial 알고리즘을 실행한 결과는 다음과 같다.

Trivial



```
C:\WINDOWS\system32\cmd.exe
3276번째 Short Read 비교
3277번째 Short Read 비교
3278번째 Short Read 비교
3279번째 Short Read 비교
3280번째 Short Read 비교
3281번째 Short Read 비교
3282번째 Short Read 비교
3283번째 Short Read 비교
3284번째 Short Read 비교
3285번째 Short Read 비교
3286번째 Short Read 비교
3287번째 Short Read 비교
3288번째 Short Read 비교
3289번째 Short Read 비교
3290번째 Short Read 비교
3291번째 Short Read 비교
3292번째 Short Read 비교
3293번째 Short Read 비교
3294번째 Short Read 비교
3295번째 Short Read 비교
3296번째 Short Read 비교
3297번째 Short Read 비교
3298번째 Short Read 비교
3299번째 Short Read 비교
3300번째 Short Read 비교
복원에 걸린 시간 : 14.9983
일치율 검사 중
일치율 : 52.58%
계속하려면 아무 키나 누르십시오...
```

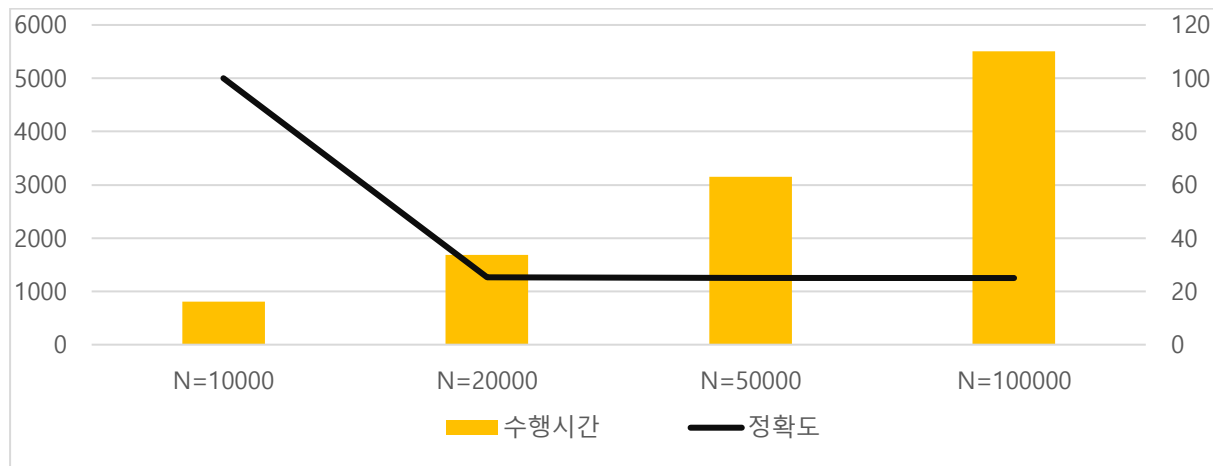
MAQ



```
C:\WINDOWS\system32\cmd.exe
Short Read 매칭... 3277
Short Read 매칭... 3278
Short Read 매칭... 3279
Short Read 매칭... 3280
Short Read 매칭... 3281
Short Read 매칭... 3282
Short Read 매칭... 3283
Short Read 매칭... 3284
Short Read 매칭... 3285
Short Read 매칭... 3286
Short Read 매칭... 3287
Short Read 매칭... 3288
Short Read 매칭... 3289
Short Read 매칭... 3290
Short Read 매칭... 3291
Short Read 매칭... 3292
Short Read 매칭... 3293
Short Read 매칭... 3294
Short Read 매칭... 3295
Short Read 매칭... 3296
Short Read 매칭... 3297
Short Read 매칭... 3298
Short Read 매칭... 3299
Short Read 매칭... 3300
실행시간 : 2229.42
MyDna 로드 중
일치율 계산 중
일치율 : 53.095%
계속하려면 아무 키나 누르십시오...
```

ShortRead의 길이와 개수, Reference 서열의 반복을 고려하여 일치율의 한계값이 대략 55~60% 정도라고 고려하였을 때 MAQ알고리즘과 Trivial알고리즘 간의 정확도 차이는 크지 않다고 볼 수 있다. 하지만 Trivial 알고리즘은 소요시간이 약 15초인 것에 반해 MAQ알고리즘은 소요시간이 약 37분 정도로 매우 길게 나타난 것을 볼 수 있다.

### 3-(4) 김준섭 :Greedy 와 De-Novo



DNA의 총 길이인 N이 클수록 수행 시간은 800에서 1690, 3149, 5507까지 급격하게 증가한다. 그리고 N의 크기가 10000보다 클 경우, 정확도는 일관되게 25%에 근사하게 나왔다. 이는 DNA sequencing이 제대로 이루어지지 못한 것이다. 하지만 N=10000의 경우 정확도가 100%로 나왔는데, 이는 DNA 크기에 비해 short read의 개수가 많아서, 즉 정보가 많아 sequencing이 제대로 이루어진 것을 알 수 있다. Trivial 알고리즘이랑 비교한 결과, Trivial 알고리즘의 수행시간은 20초 이내, 정확도는 55~65%로 수행시간은 짧지만 정확도에 한계를 보였다. 반면 Greedy 알고리즘의 경우 N=50000이나 N=100000의 경우에도 short read의 길이나 개수를 증가시키면 정확도를 높일 수 있지만, 수행시간이 기하급수적으로 커져서 또한 한계가 있다.