Program do mapowania odczytów

Algorytmy analizy danych genomicznych

Younginn Park

FM indeks

“Downsamplowana” (domyślnie co 50 nt):

BWT – transformata Burrowsa-Wheelera

SA – tablica sufiksowa

oraz

first – zliczenie alfabetu (pierwsza kolumna BWM)

T – oryginalny tekst (sekwencja)…

Konstrukcja FM indeksu

- Kod z laboratorium (zmodyfikowany do projektu):
  - **SA** – Karkkainen-Sanders
  - **bwtFromSa** – funkcja z laboratorium
  - **FmIndex** – klasa z laboratorium
  - **smithWaterman** i **traceback_with_indices** – lokalne uliniawianie z DP i dekodowanie indeksów uliniowienia
  - **editCost** – koszt edycji (1 – *match*, -1 *mismatch*)

```
class FmIndexApprox(FmIndex):
    """Subclass with additional functionalities for approximate matching"""
    def __init__(self, t, cpIval=50, ssaIval=50):
        super().__init__(t, cpIval, ssaIval)

    def generate_kmers(self, query_pattern, k):
        """Generate k-mers of length k"""
        return [query_pattern[i:i+k] for i in range(len(query_pattern) - k + 1)]

    def generate_spaced_kmers(self, query_pattern, k, ki):
        """ Generate k-mers of length k with interval ki, return k-mers and start
indices """
        kmers = [query_pattern[i:i+k] for i in range(0, len(query_pattern) - k + 1,
ki)]
        indices = list(range(0, len(query_pattern) - k + 1, ki))
        return kmers, indices

class FmIndexWithT(FmIndexApprox):
    def __init__(self, t, cpIval=50, ssaIval=50):
        super().__init__(t, cpIval, ssaIval)
        self.t = t
```

Mapowanie – seed and extend

- **Seed** – k-mery (k=23 i ki=17 najlepiej działały czasowo)
- Szybkie wyszukiwanie dokładnych trafień (*exact match*) dla seedów
- Znajdywanie potencjalnego zakresu mapowania odczytu – konsensus na podstawie znalezionych dokładnych seedów

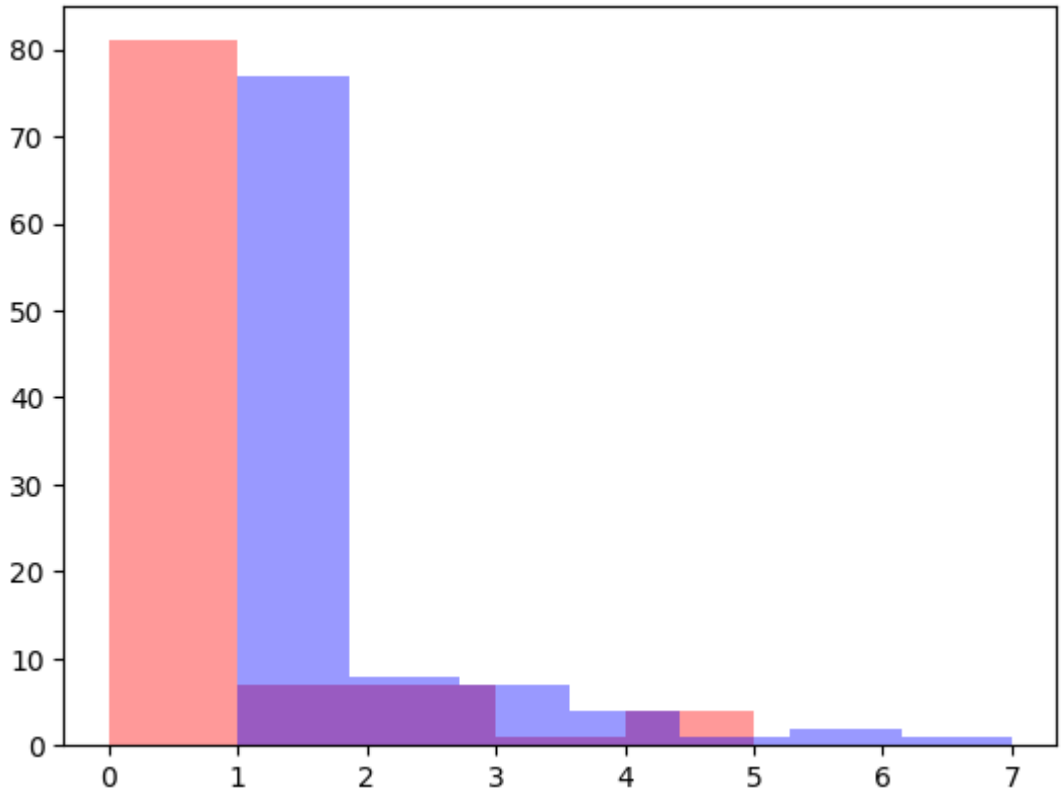
- **Extend** – lokalne uliniawianie fragmentów (początku i końca potencjalnego zakresu) w celu dokładniejszego zlokalizowania końcówek

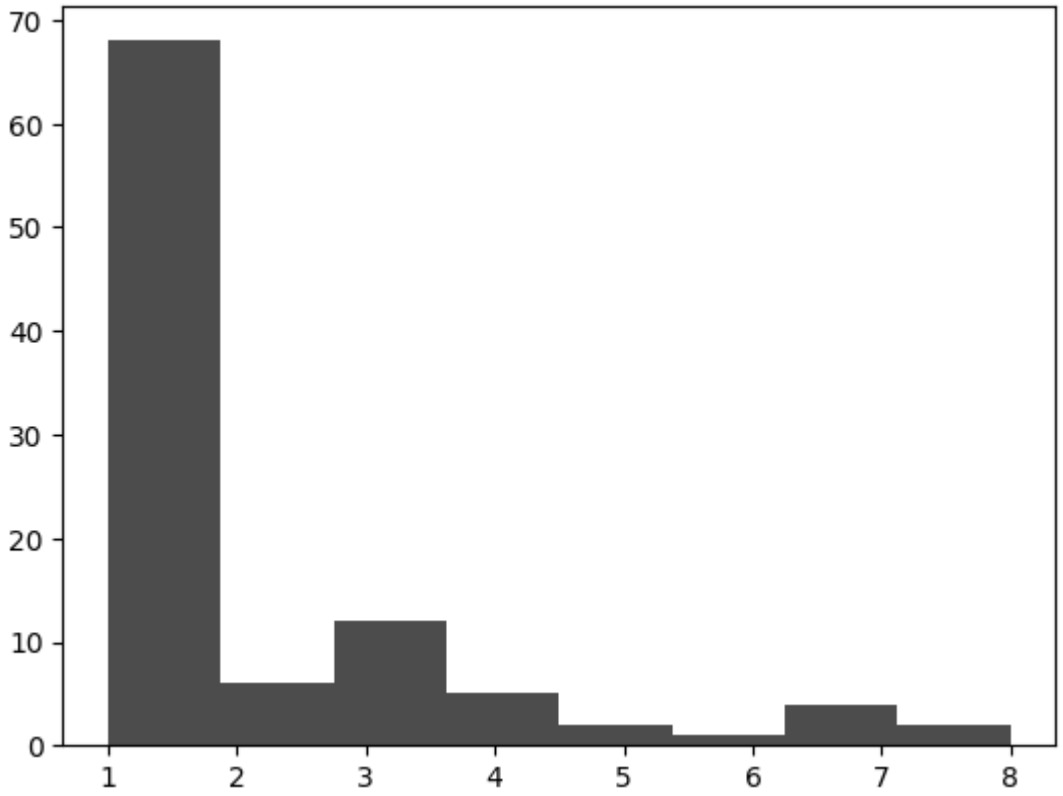
```
s = editCost
mapping = {}

not_mapped = 0 # how many reads were not mapped
num_reads = 0

for read in reads:
    num_reads += 1

    klen = 23
    kint = 17
    rlen = len(read.seq)

    kmers, inds = fm_index.generate_spaced_kmers(read, k=klen, ki=kint)

    kmer_occs = {}
    for it in range(len(kmers)):
        kmer = kmers[it]
        i = inds[it]
        kmer_occs[i] = fm_index.occurrences(kmer.seq)

    found = sum(map(len, kmer_occs.values()))
    if not found:
        # If no exact match for any of the k-mers is found
        not_mapped += 1
        continue

    # "Mapping"
    proposals = {}
    proposals_starts = {} # starting positions of kmers involved in a proposal
```

```
found = sum(map(len, kmer_occs.values()))
if not found:
    # If no exact match for any of the k-mers is found
    not_mapped += 1
    continue

# "Mapping"
proposals = {}
proposals_starts = {} # starting positions of kmers involved in a proposal
for kmer, match_list in kmer_occs.items():
    for match_position in match_list:

        pstart = match_position - kmer
        pstop = match_position + (rlen - kmer)

        if pstart < 0 or pstop > fm_index.slen:
            continue

        proposal = (pstart, pstop)
        if (pstart, pstop) in proposals.keys():
            proposals[proposal] += 1
            proposals_starts[proposal].append(match_position)
        else:
            proposals[proposal] = 1
            proposals_starts[proposal] = [match_position]

# Edge refinement
# best proposal - the one with the most confirmation
kmer_starts = proposals_starts[max(proposals, key=proposals.get)]
naive_proposal = max(proposals, key=proposals.get)

# indices of T and read that will be aligned
safety = 5
```

```
# Edge refinement
# best proposal - the one with the most confirmation
kmer_starts = proposals_starts[max(proposals, key=proposals.get)]
naive_proposal = max(proposals, key=proposals.get)

# indices of T and read that will be aligned
safety = 5
tl_start = max(naive_proposal[0] - safety, 0)
tl_end = min(kmer_starts)
tr_start = max(kmer_starts) + klen
tr_end = min(naive_proposal[1] + safety, fm_index.slen)

pl_start = 0
pl_end = tl_end - tl_start + safety
pr_start = tr_start - tl_start - safety
pr_end = rlen

tl = fm_index.t[tl_start : tl_end]
pl = read.seq[pl_start : pl_end]
tr = fm_index.t[tr_start : tr_end]
pr = read.seq[pr_start : pr_end]

tl_inds, pl_inds = traceback_with_indices(tl, pl, s)
tr_inds, pr_inds = traceback_with_indices(tr, pr, s)

new_start = tl_inds[0] + tl_start
new_end = min(tr_inds[1] + tr_start, fm_index.slen)

new_proposal = (new_start, new_end)

mapping[read.id] = new_proposal
```

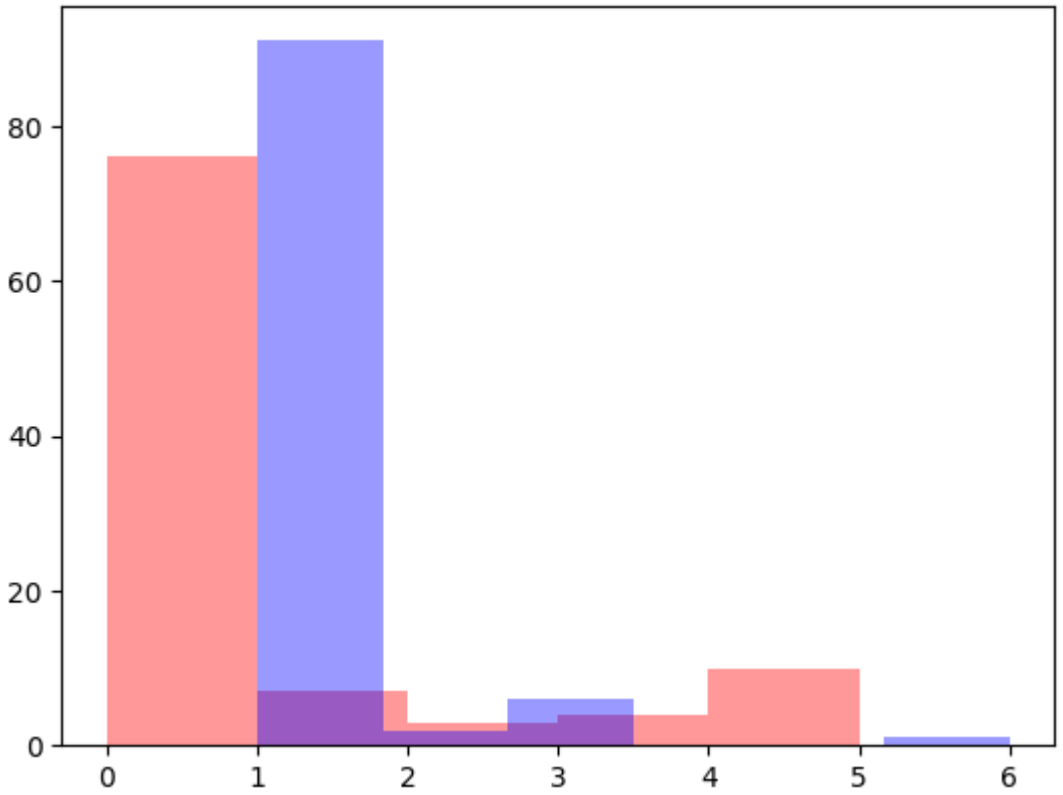
Wyniki – dokładność dla reads2

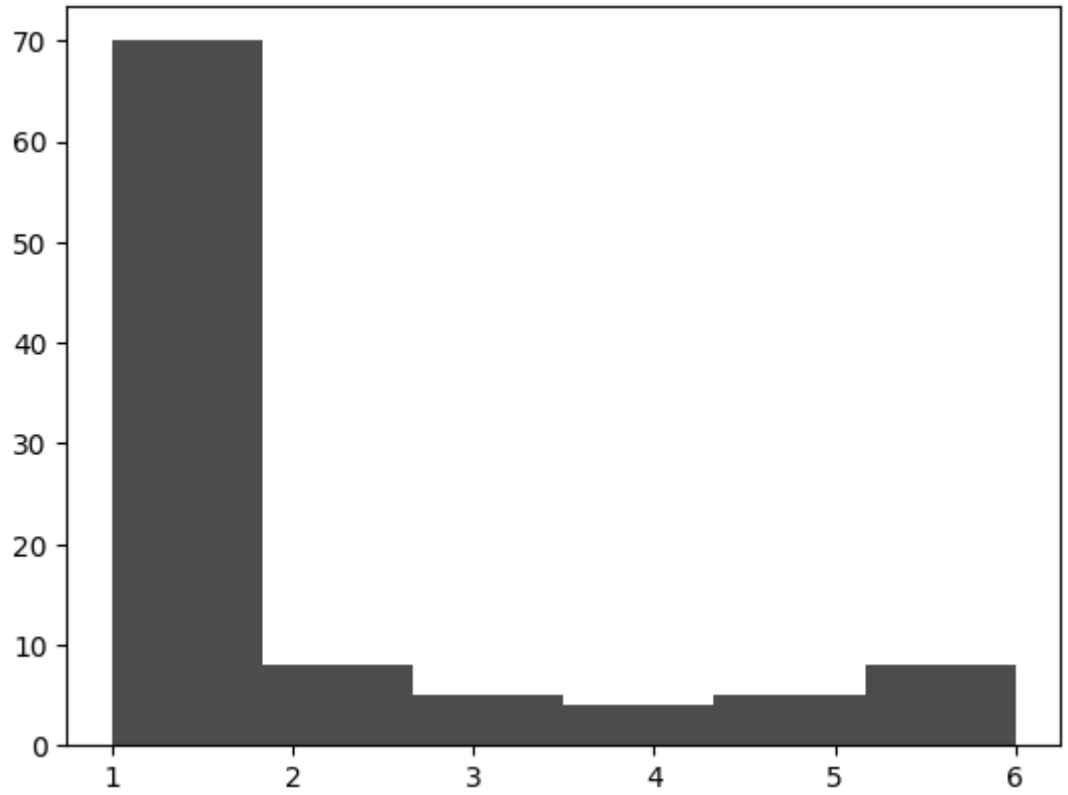




Zakładając, że przykładowe wyniki z moodle’a są “ground truth’em” – różnica pozycji początkowej odczytu (czerwone po lewej) i końcowej (niebieskie po lewej) i całkowita różnica (po prawej)

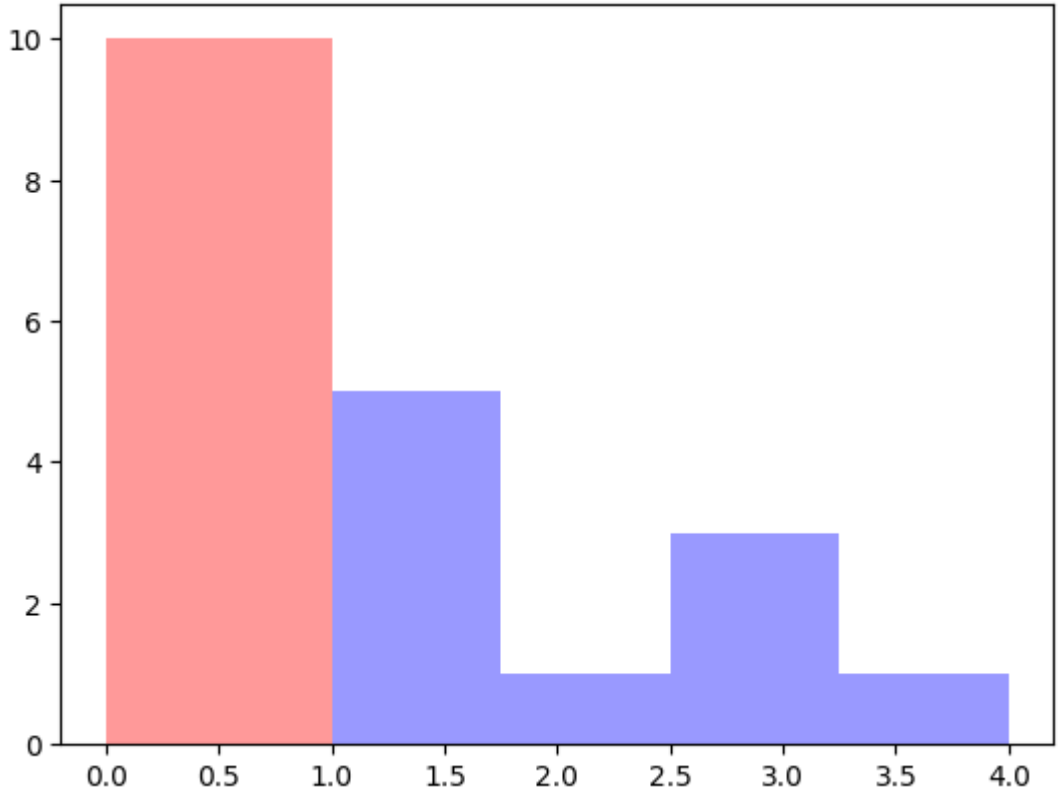
Wyniki dla reads1

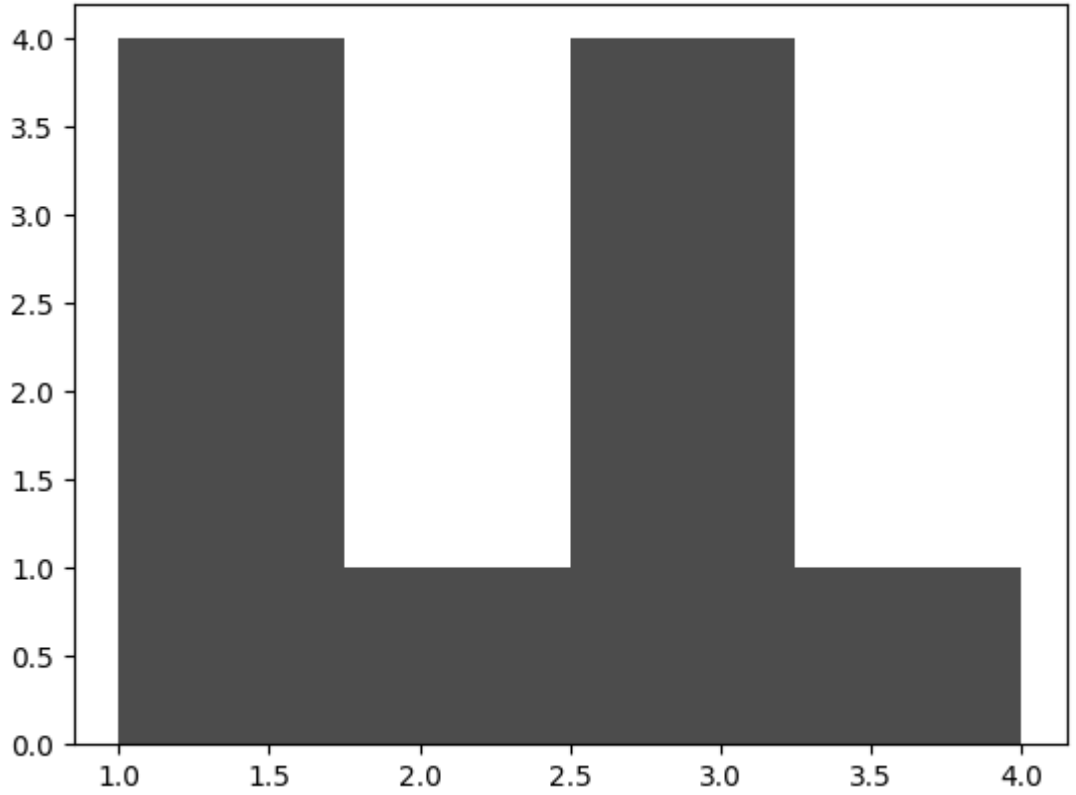




Zakładając, że przykładowe wyniki z moodle’a są “ground truth’em” – różnica pozycji początkowej odczytu (czerwone po lewej) i końcowej (niebieskie po lewej) i całkowita różnica (po prawej)

Wyniki dla reads0





Zakładając, że przykładowe wyniki z moodle’a są “ground truth’em” – różnica pozycji początkowej odczytu (czerwone po lewej) i końcowej (niebieskie po lewej) i całkowita różnica (po prawej)

Czas i pamięć

- Spełnia minimalne wymogi czasowe i pamięciowe (2 + r/10 minut oraz <1Gb na serwerze *students*)
- Przy k=23, ki=17, mapuje wszystkie odczyty (przykładowe z moodle’a)
- Czasowo mapowanie daje <1s na 1 odczyt