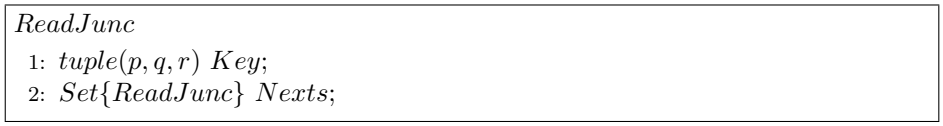

Figure 1: Data structure *ReadJunc* for storing alignments.

1 Data structure

ReadJunc as shown in Figure-1 is a recursive data structure used to store the alignments found by $Find_All_Juncs$. It has two components, the *key* and a set containing its descendants. The *key* is a tuple of three elements, $p, qandr$, where (p, q) define the starting and ending coordinates of the junction in the reference genome, respectively. The p component is 0-based and the q component is 1-based. Component r is the size of the left anchor of the junction.

```
Find_All_Junc(R, Φ = (G, l, L_min, k, D_1..D_2))
 1: Min_Juncs = ∞;
 2: ReadJunc H;
 3: H.Key = (0, 0, 0);
 4: H.Nexts = Seek_Junc(R, 1, Φ, {[1, |G|]}, 0);
 5: Transcripts = ∅;
 6: Enum_Transcripts(H, T, 0, Min_Junc, Transcripts)
 7: Remove_Duplicates(Transcripts);
```

```
Seek_Junc(S, a, Φ, X, L)
 1: Juncs = ∅; T_1 = T_2 = A = ∅;
 2: if a = |S| then
 3:    Min_Junc = L;
 4:    return Juncs;
 5: end if
 6: A = Extend(S[a, min(a + l, |S|)], X, Φ)
 7: if A ≠ ∅ then
 8:    Juncs = Seek_Junc(S, min(a + l, |S|), Φ, A, L);
 9: end if
10: if a > 1 and |S| − a > l then
11:    if X contains only genomic locations then
12:       X = X − a;
13:    else
14:       X = ∅
15:    end if
16:    if L < Min_Junc then
17:       T_1 = Find_Single_Junc(S[1, min(a + 2l − 1, |S|)], Φ, X);
18:       for (p, q, r) ∈ T_1 do
19:          ReadJuncH;
20:          ReadJuncH.key = (p, q, r);
21:          H.Nexts = Seek_Junc(S[r + 1, |S|], 1, Φ, {q}, L + 1);
22:          Hits.add(H);
23:       end for
24:    end if
25:    if L + 1 < Min_Junc and L_min < 2l then
26:       for i = 1, .., 2l − L_min − 1 step L_min − l do
27:          T_2 = T_2 ∪ Find_Single_Junc(S[1, min(a + (L_min − l) + i, |S|)], Φ, X);
28:       end for
29:       for (p, q, r) ∈ T_2 do
30:          ReadJuncH;
31:          ReadJuncH.key = (p, q, r);
32:          H.Nexts = Seek_Junc(S[r + 1, |S|], 1, Φ, {q}, L + 1);
33:          Hits.add(H);
34:       end for
35:    end if
36: end if
37: if A = ∅ and T_1 = ∅ and T_2 = ∅ then
38:    return NULL;
39: end if
40: return Juncs;
```


Figure 2: Algorithms *Find_All_Junc* and *Seek_Junc*.

```
Enum_Transcripts(H, T, L, Min_Junc, Transcripts)

 1: if L > Min_Junc or H.Nexts = NULL then
 2:    return
 3: end if
 4: if H.Nexts = ∅ then
 5:    if L = Min_Junc then
 6:       store T in Transcripts;
 7:    end if
 8:    return  ;
 9: end if
10: for each J in H.Nexts do
11:    T[L] = J.Key;
12:    Enum_Transcripts(J, T, L + 1, Min_Junc, Transcripts);
13: end for
```

Figure 3: Algorithm *Enum_Transcripts*.