

**Finding k-cliques using Backtracking with cutoffs pruning**

```

CliqueBT (Vector A, j) {
  // If j is equal to size of clique, k, then A is k-clique in the graph
  if (j == sizeClique) then do {
    numClique++
    return
  }
  else {
    j = j + 1
    if (j <= sizeClique) then do {
      // Let Sj is the set of all candidate vectors for j-clique
      Sj = getCandidates (A)
    }
    if (Sj is NOT empty) {
      // For each candidate vector in Sj, recursively do backtracking for k-clique
      for (each candidate vector aj in Sj) do
        CliqueBT (aj, j)
    }
  }
}

```

/\* Return a set of candidates, S<sub>q</sub>, for q-clique. Each candidate is a vector in which a newly added vertex must be greater than the last vertex in the given vector, A, and must be connected to all vertices in A. Note that the returning candidate vectors are extended from the given vector A by adding only one additional vertex each time this method is called.  
\*/

```

getCandidates (Vector A) {
  Let candidatesList be Sj, the set of all candidate vectors for q-clique

  // If A is empty, let sj be a vector with each singleton node in the graph.
  if (A is empty) then do
    candidatesList.add (each node in the graph)
  else {
    // Permutate all candidate vectors, satisfying the property of sj
    for (all elements j > A.lastElement) do {
      allConnected := true
      // Check if vertex 'j' is adjacent to all vertices in A
      for (every neighbor i of j) do {
        if ( i and j is NOT connected) then do {
          // Cutoff occured in pruning - fails to meet the property of A
          allConnected := false
          break
        }
      }
      if (allConnected == true) then do
        candidatesList.add (sj)
    }
  }
  return candidatesList
}

```