**histogram pruning 的代码段和beam pruning 的代码段**

Part1. ProcessEmitting()

histogram pruning: (tok->tot\_cost <= cur\_cutoff) 支路部分

beam pruning: (tot\_cost >= next\_cutoff) 支路部分

line774-799

if (tok->tot\_cost <= cur\_cutoff) {  
 for (fst::ArcIterator<FST> aiter(\*fst\_, state);  
 !aiter.Done();  
 aiter.Next()) {  
 const Arc &arc = aiter.Value();  
 if (arc.ilabel != 0) { // propagate..  
 BaseFloat ac\_cost = cost\_offset -  
 decodable->LogLikelihood(frame, arc.ilabel),  
 graph\_cost = arc.weight.Value(),  
 cur\_cost = tok->tot\_cost,  
 tot\_cost = cur\_cost + ac\_cost + graph\_cost;  
 if (tot\_cost >= next\_cutoff) continue;  
 else if (tot\_cost + adaptive\_beam < next\_cutoff)  
 next\_cutoff = tot\_cost + adaptive\_beam; // prune by best current token  
 // Note: the frame indexes into active\_toks\_ are one-based,  
 // hence the + 1.  
 Elem \*e\_next = FindOrAddToken(arc.nextstate,  
 frame + 1, tot\_cost, tok, NULL);  
 // NULL: no change indicator needed  
  
 // Add ForwardLink from tok to next\_tok (put on head of list tok->links)  
 tok->links = new ForwardLinkT(e\_next->val, arc.ilabel, arc.olabel,  
 graph\_cost, ac\_cost, tok->links);  
 }  
 } // for all arcs  
}

Part2. ProcessNonemitting()

beam pruning: (cur\_cost >= cutoff) 支路部分

histogram pruning: (tot\_cost < cutoff) 支路部分

line:855-885

if (cur\_cost >= cutoff) // Don't bother processing successors.  
 continue;  
// If "tok" has any existing forward links, delete them,  
// because we're about to regenerate them. This is a kind  
// of non-optimality (remember, this is the simple decoder),  
// but since most states are emitting it's not a huge issue.  
DeleteForwardLinks(tok); // necessary when re-visiting  
tok->links = NULL;  
for (fst::ArcIterator<FST> aiter(\*fst\_, state);  
 !aiter.Done();  
 aiter.Next()) {  
 const Arc &arc = aiter.Value();  
 if (arc.ilabel == 0) { // propagate nonemitting only...  
 BaseFloat graph\_cost = arc.weight.Value(),  
 tot\_cost = cur\_cost + graph\_cost;  
 if (tot\_cost < cutoff) {  
 bool changed;  
  
 Elem \*e\_new = FindOrAddToken(arc.nextstate, frame + 1, tot\_cost,  
 tok, &changed);  
  
 tok->links = new ForwardLinkT(e\_new->val, 0, arc.olabel,  
 graph\_cost, 0, tok->links);  
  
 // "changed" tells us whether the new token has a different  
 // cost from before, or is new [if so, add into queue].  
 if (changed && fst\_->NumInputEpsilons(arc.nextstate) != 0)  
 queue\_.push\_back(e\_new);  
 }  
 }  
} // for all arcs