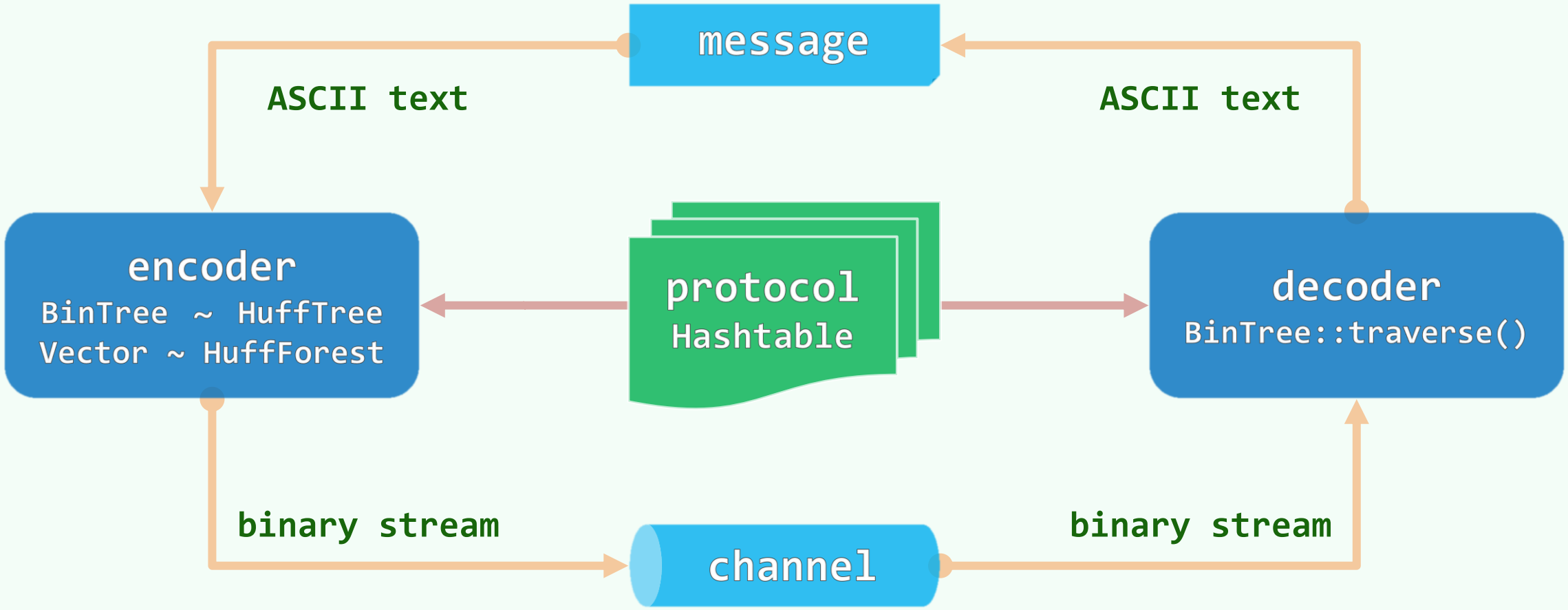
二叉树

Huffman编码树：算法实现

05-13

树形建筑也出现了，看上去规模与地球上的差不多，
只是挂在树上的建筑叶子更为密集。

邓俊辉
deng@tsinghua.edu.cn


message
ASCII text
ASCII text
encoder
BinTree ~ HuffTree
Vector ~ HuffForest
protocol
Hashtable
decoder
BinTree::traverse()
binary stream
binary stream
channel

Huffman（超）字符

```
#define  N_CHAR  (0x80 - 0x20) //仅以可打印字符为例

struct HuffChar { //Huffman（超）字符

   char ch; int weight; //字符、频率

   HuffChar ( char c = '^', int w = 0 ) : ch ( c ), weight ( w ) {};

   bool operator<  ( HuffChar const& hc ) { return weight >  hc.weight; } //比较器

   bool operator== ( HuffChar const& hc ) { return weight == hc.weight; } //判等器

};
```

Huffman树与森林

- Huffman（子）树

```
using HuffTree = BinTree< HuffChar >;
```

- Huffman森林

```
using HuffForest = List< HuffTree* >;
```

- 待日后掌握了更多数据结构之后，可改用更为高效的方式，比如：

```
using HuffForest = PQ_List< HuffTree* >; //基于列表的优先级队列
using HuffForest = PQ_ComplHeap< HuffTree* >; //完全二叉堆
using HuffForest = PQ_LeftHeap< HuffTree* >; //左式堆
```

- 得益于已定义的统一接口，支撑Huffman算法的这些底层数据结构可直接彼此替换

构造编码树

```
HuffTree* generateTree( HuffForest * forest ) { //Huffman编码算法
   while ( 1 < forest->size() ) { //反复迭代，直至森林中仅含一棵树
      HuffTree *T1 = minHChar( forest ), *T2 = minHChar( forest );
      HuffTree *S = new HuffTree(); //创建新树，准备合并T1和T2
      S->insert( HuffChar( '^', //根节点权重，取作T1与T2之和
         T1->root()->data.weight + T2->root()->data.weight ) );
      S->attach( T1, S->root() ); S->attach( S->root(), T2 );
      forest->insertAsLast( S ); //T1与T2合并后，重新插回森林
   } //assert: 循环结束时，森林中唯一的那棵树即Huffman编码树
   return forest->first()->data; //故直接返回之
}
```

查找最小超字符

❖ Huffman编码的整体效率，直接决定于minHChar()的效率

```
HuffTree* minHChar( HuffForest * forest ) { //此版本仅达到O(n)，故整体为O(n²)
   ListNodePosi( HuffTree* ) p = forest->first(); //从首节点出发
   ListNodePosi( HuffTree* ) minChar = p; //记录最小树的位置及其
   int minWeight = p->data->root()->data.weight; //对应的权重
   while ( forest->valid( p = p->succ ) ) //遍历所有节点
      if( minWeight > p->data->root()->data.weight ) { //如必要，则
         minWeight = p->data->root()->data.weight; minChar = p; //更新记录
      }
   return forest->remove( minChar ); //从森林中摘除该树，并返回
}
```

构造编码表

```
#include "Hashtable.h" //用HashTable（第11章）实现
using HuffTable = Hashtable< char, char* >; //Huffman编码表
static void generateCT //通过遍历获取各字符的编码
   ( Bitmap* code, int length, HuffTable* table, BinNodePosi(HuffChar) v ) {
   if ( IsLeaf( * v ) ) //若是叶节点（还有多种方法可以判断）
      { table->put( v->data.ch, code->bits2string( length ) ); return; }
   if ( HasLChild( * v ) ) //Left = 0，深入遍历
      { code->clear(length); generateCT( code, length + 1, table, v->lc ); }
   if ( HasRChild( * v ) ) //Right = 1
      { code->set(length);   generateCT( code, length + 1, table, v->rc ); }
} //总体O(n)
```