Packed CBOR

draft-ietf-cbor-packed-00

Carsten Bormann, CBOR interim, 2020-10-28

JSON, CBOR: Coding efficiency

- CBOR can be more efficient than JSON, in particular if the data model is specifically designed for CBOR (e.g., integer labels in maps)
- Simply encoding JSON data in CBOR reaps less gain
- Significant redundancy often remains
  - Can be removed by, e.g. DEFLATE (RFC 1951)
  - Compression requires decompression before use, though
- Alternative: Exploiting structure and prefix sharing by “**Packing**”
  - CBOR data item can be used while remaining packed

Item Sharing
(née Structure Sharing)

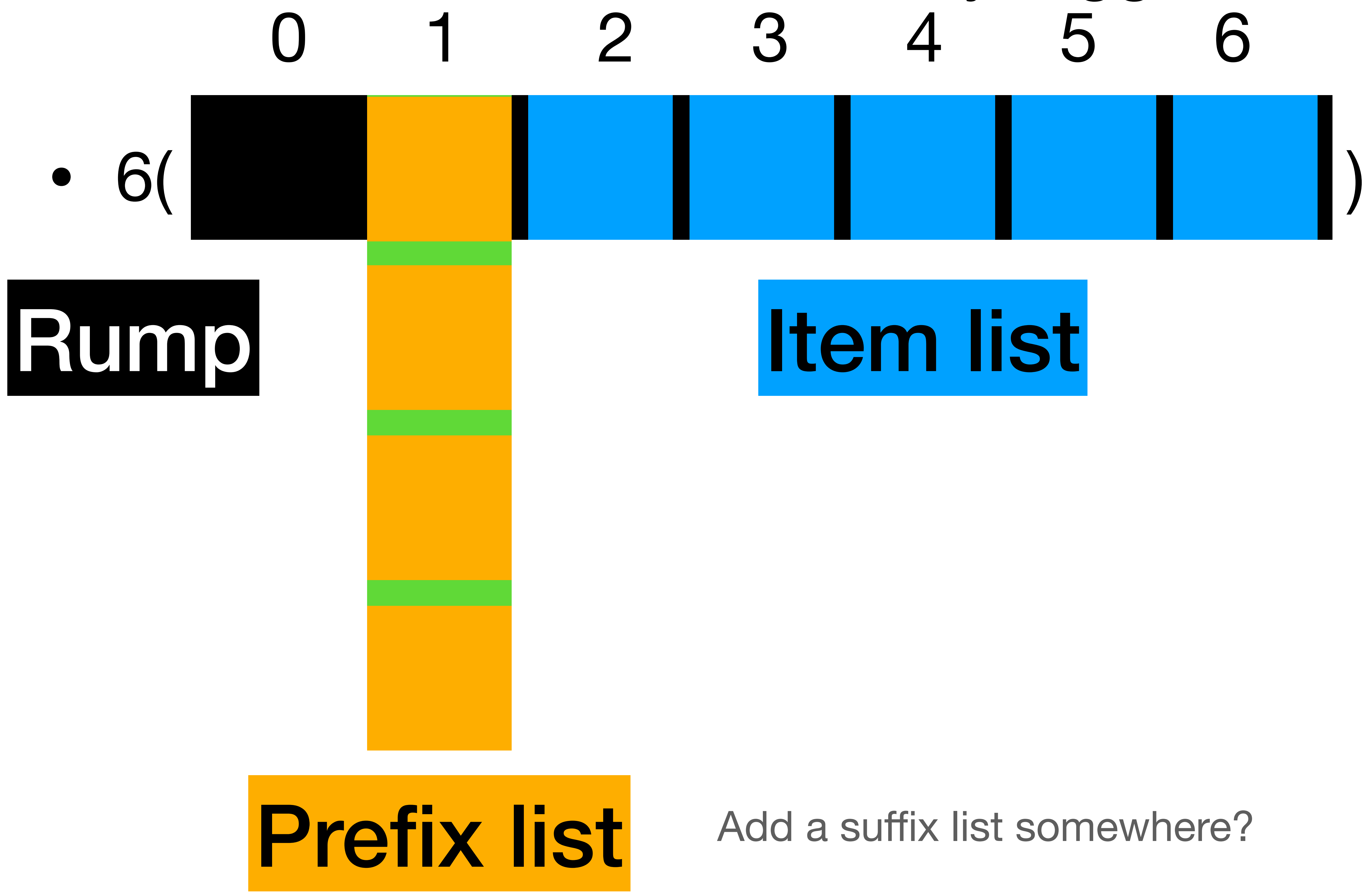
- Many data items nested in a larger data item repeat
  - E.g., strings used for labels or enums
- Idea: Provide one copy of repeated item and **share** it
- Item is
  - put into an item sharing array,
  - referenced in the places where a copy is needed

Prefix/Suffix (Affix) Sharing

- data items often share a **prefix** or a **suffix (an affix)**
  - E.g., initial parts of URIs are often similar
- Idea: Provide one copy of repeated affix and **share** it
- Shared -fix is
  - put into a prefix array or suffix array,
  - referenced in the places where a copy is needed
- –00 only defines this for (byte and text) strings; extend to arrays and maps

Structure of packed CBOR (-00)

- Packed data item is an array tagged with tag 6:
- 6(…)



Add a suffix list somewhere?

- Rump can reference shared items; shared items can, too (yes, needs loop detection)
- Items can use a prefix (identified by a tag) plus a supplied suffix, or a suffix plus a supplied prefix

Elements of a generalization

- Cbor-packed has two major components:
  - Referents that can be used in place of a data item
    - Need to use a namespace to identify what is being referenced
    - Short (= early) names are good
    - Items/prefixes/suffixes don’t mix much ➔ separate namespaces are good
  - Tables that populate the namespaces
    - –00 has two (item, prefix), self-contained
    - Proposal: add dictionaries to share (!) the populations
      - From outer structure in CBOR data item
      - From some registered or (hash-)identified space

–00: efficient Item and Prefix references

- Item references: 16 simple values (1+0),
  one single-byte Tag ➔ 48+512+131072 (1+1, 1+2, 1+4)
- Prefix references: Reuse tag; use more tags (32+4096+268435456)
  Do the same (but not necessarily the same sizes) separately for suffix
- Total reservation: 4/7 simple values, 1 1+0 tag (1/24), 1/8 1+1, 1/16 1+2, …
- Worth it if we think this will be a widely used part of CBOR
- Could be less agressive and less efficient, but why?

How to build tables

- Position in table is relevant
  - At least within a bucket:
    - Items: 16, 48, 512, 131072
    - Prefixes/Suffixes: 32, 4096, 268435456
- Combining imported and locally defined tables
  - Use imported only?   Use locally defined only (= –00)?
  - When using both, sequence becomes important when a bucket overflows

How to reference dictionaries (external tables)

- Referencing (and table building!) scheme could be orthogonal to packing scheme
- URIs: Identify + locate
- Hashes: Identify only
- (IANA-)Registered dictionaries: Identify; locate if known

Strawman: add after end of local table

Building tables from multiple sources

- Per-bucket structure (4i+3p+3s buckets total!); add at end
- Overflow goes to end of next higher bucket of same type that has space
- Requires a defined sequence of subtables
  - Local, then dictionaries in defined order?
  - Define sequence in structure that provides values/references?