Data Structures

Introduction – 1/4

- A language processor makes frequent use of the search operation over its data structures

- Data structures used in language processing can be classified on basis of the following criteria
  - **Nature** of a data structure :- whether linear or non linear
  - **Purpose** of a data structure :- whether search or allocation
  - **Lifetime** of a data structure :- whether used during language processing or during target program execution

Introduction – 2/4

- **Linear data structure**
  - Consists of linear arrangement of elements in memory
  - Requires contiguous area of memory
  - Designer is forced to overestimate the memory requirements
  - Leads to wastage of memory

- **Non linear data structure**
  - Accessed using pointers
  - Elements do not occupy contiguous area of memory
  - Leads to lower search efficiency

Introduction – 3/4

- **Search data structure**
  - Used during language processing to maintain attribute information concerning different entities in SP
  - Entry for entity is created only once
  - Searched for large no of times
  - Search efficiency is very important
  - Constitute various tables of information

Introduction – 4/4

- **Allocation data structure**
  - Address of memory area allocated to entity is known to user of that entity
  - No search operations are conducted
  - Speed of allocation or deallocation and efficiency of memory utilization are important
  - Used to handle programs with nested structures
- TP rarely uses search data structures

Search data structures

- Set of entries, each entry accommodating the information concerning one entity
- Each entry has a key field which forms basis of search
- **Key field is symbol field containing name of entity**

Entry formats – 1/2

- Each entry in search structure is a set of fields
- Entry consists of two parts
  - **Fixed part**
  - **Variant part**
- Each part consists of set of fields
- Fields of fixed part exist in each entry of search structure

Entry formats – 2/2

- Value in tag field of fixed part determines the information to be stored in variant part of entry
- Fixed and variant part may be nested (i.e. variant part may itself consist of fixed and variant parts)

Fixed and Variable length entries – 1/3

- An entry may be declared as a **record or a structure** of language in which the language processor is being implemented

- In **fixed length entry format**, each record is defined to consist of the following fields
  - **Fields in the fixed part of entry**
  - **U_{vi} SF_{vi}, i.e. set of fields in all variant parts of entry**

- Records in search structure have identical format which enables use of homogeneous linear data structures like arrays

Fixed and Variable length entries – 2/3

- Use of linear search organizations enables use of efficient search procedures

- In **variable length entry format**, record consists of following fields
  - **Fields in fixed part of entry, including tag field**
  - **$\{ f_j \mid f_j \in SF_{vj}$ if tag $= v_j \}$**

- No memory wastage occurs

Fixed and Variable length entries – 3/3

- When variable length entry format is used, search method may require knowledge of length of entry

- Hence, record might consists of fields
  - A length field
  - Fields in fixed part of entry, including tag field
  - $\{ f_j \mid f_j \in SF_{vj} \text{ if tag} = v_j \}$

- Depict as follows

length	***entry***

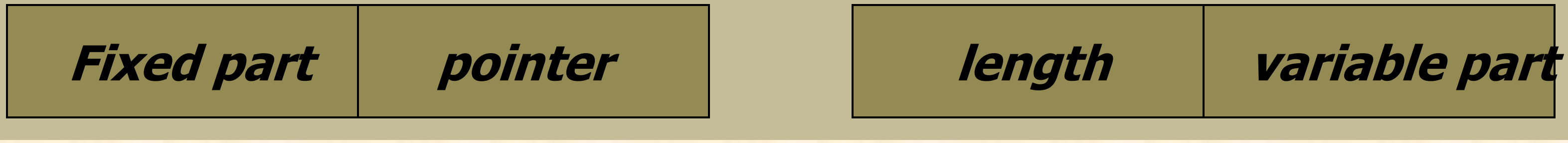
Hybrid entry formats – 1/2

- Compromise between fixed and variable entry formats to combine access efficiency of fixed entry format with memory efficiency of variable entry format
- Split into two halves
  - **Fixed part**
  - **Variant part**
- Pointer is added to fixed part
- It points to variable part of entry

Hybrid entry formats – 2/2

- Fixed part of all entries are organized into an efficient search structure eg. Linear structure
- Variable part does not need to be located through a search
- Hence it is put into an allocation data structure

- Hybrid entry format is depicted below



Fixed part ***pointer*** ***length*** ***variable part***

Operations on search structures

- **Add** :- Add entry of a symbol
- **Search** :- Search and locate entry of a symbol
- **Delete** :- Delete entry of a symbol

- Entry for a symbol is created only once, but can be searched many no of times
- Deletion operation is not common

Generic search procedure – 1/2

1. **Make a prediction** concerning entry of search data structure which symbol s may be occupying. Let this entry be **e**

2. Let $\mathbf{s_e}$ be symbol occupying eth entry. **Compare s with** $\mathbf{s_e}$**. Exit with success if two match**

3. Repeat steps 1 and 2 till it can be concluded that the symbol does not exist in search data structure

Generic search procedure – 2/2

- Nature of prediction varies with organization of search data structure
- **Each comparison is called a probe**
- **Efficiency is determined by no of probes performed in search procedure**

- Following notation to represent no of probes in a search
  - p_s :- no of probes in successful search
  - p_u :- no of probes in unsuccessful search

Other Topics

- Table organization
- Sequential search organization
- Binary search organization
- Hash table organization
  - Hashing functions
  - Collision handling methods
- Linked list and tree structured organization
- Allocation data structures
  - Stacks
  - Heaps