Applied Programming and Computer Science, DD2325/appcs14

PODF, Programmering och datalogi för fysiker, DA7011

Lecture 6, Huffman coding and hashing

A. Maki, C. Edlund
December 2014

Tree

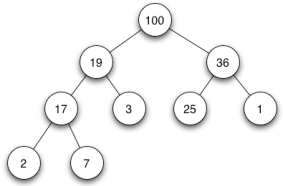
When are trees used?

- Binary heap / Heap sort
- Data compression
- Databases
- File system
- Compiler uses syntax trees
- Chess program
- Decision trees
- ...

Binary heap

A binary heap is a binary tree with two additional constraints

- Heap property: All nodes are greater(less) than or equal to each of its children
- Shape properry: All levels of the tree except the deepest are filly filled (complete binary tree)



Siblings in a heap can be interchanged unless doing so violates the constraints.

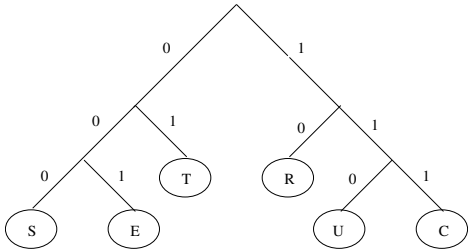
How can we build a heap from a set of data?
Given a binary heap, Heap sort algorithm is straightforward.

Compression, Huffman code

Every character is coded binary.
The common characters have short code. Less common, longer code. Statistically decided.

Example `TREESTRUCTURE`

The characters T, R and E are most common and therefore have short code. The tree structure for the example:



Compression, Huffman code (cont.)

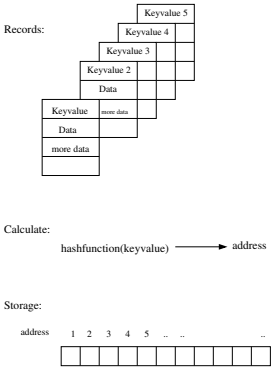
How can `TREESTRUCTURE` be represented in binary code?
How many bits are required in total?

Compare the length with ASCII (using a byte) which ends up with:
13*8 = 104 bits

Compression is used when zipping files, to compress pictures/movies (files) ...

Hashing

Hashing is a way to obtain both efficient memory use and effective retrieval.
The keyvalue of a record is used to calculate the address where the record is to be stored.



Hash function

The requirements one can have on the hash function:

- must calculate an address within the legal address space.
- addresses should be evenly distributed in the legal address space.
- easy to calculate an address.

The hash function can result in the same address for different keys.
The collisions can be dealt with `open` or `closed` address hashing.

Open and closed address hashing

Open address hashing:

Linear probing.
If the keyvalue has given the address to an already occupied space the next adress space is considered.

Closed address hashing:

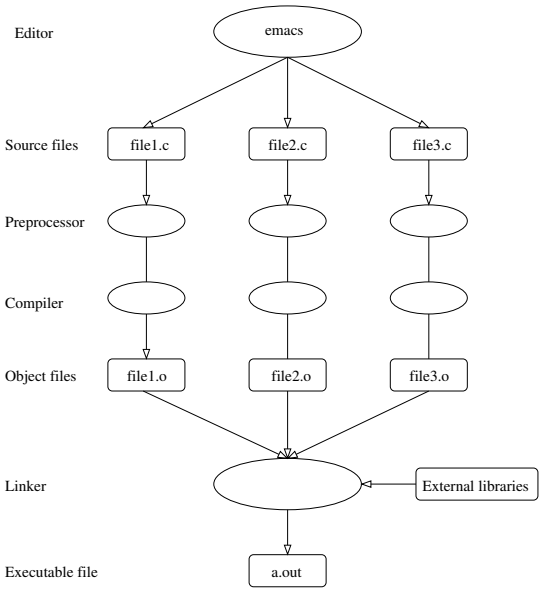
Chained hashing.
Every entry in the storage is a list.
All records with the same address ends up in the same list.

From problem to executable file

- **Thinking** A problem is formulated, algorithms devised and data structures chosen.
- **Programming** The C program is written using an editor (emacs, xemacs).
- **Preprocessing** Preprocessor directives are interpreted and a preprocessed C file generated (gcc -E, cc -E, cpp).
- **Compiling** The preprocessed source code is translated to object code (machine instructions) (gcc -c, cc -c).
- **Linking** The object code is linked with external libraries to produce an executable (gcc, cc, ld).
- **Debugging** Make sure the program works as expected. Correct possible mistakes (gdb).

The first step is the most important! By considering the problem carefully the debugging time can be significantly reduced.

Stack



Compiling & linking (cont.)

At NADA there are two different C compilers, `cc` (Sun) and `gcc` (GNU). Both can be used for preprocessing, compiling and linking small programs in a single command.

`gcc file.c -o runme`
generates an executable `runme` from the source code in `file.c`.

The object files are linked with the most common libraries (`stdio`, `stdlib` and possibly a few more).

If compiling a program is complicated (several different files, many options, different programming languages, . . .) it is convenient to use *Makefiles*.

Compiler & linker options

In some cases additional information must be supplied for the compiler to generate an executable. These are given as *command line options*. There are also some options useful for debugging and optimizing.

Option		
`gcc`	`cc`	**Effect**
`-c`	`-c`	Compile source files but do not link
`-l`*lib*	`-l`*lib*	Link with library *lib*
`-L`*dir*	`-L`*dir*	Add *dir* to library search path
`-I`*dir*	`-I`*dir*	Add *dir* to include file search path
`-o`*tgt*	`-o`*tgt*	Name executable *tgt* (default:`a.out`)
`-O2`	`-fast`	Compile with optimization
`-Wall`	`-v`	Print “extra” warning messages

For a more complete list of options, give the command:
`man gcc` (or `man cc`) at the prompt.

Compiling – examples

Compile `program.c` and generate executable `target` with optimization,

```
> gcc program.c -O2 -o target
```

Same example, but linked with math library

```
> gcc program.c -O2 -o target -lm
```

Compile files `main.c` and `function.c` and link to obtain executable `runme`

```
> cc -c main.c
> cc -c function.c
> cc function.o main.o -o runme -lm
```

User-defined include files and libraries,

```
> cc program.c -L~/mylibraries -I~/myincludefiles -lmylib
```

(">" is used to indicate that the command is given at the prompt.)

stack.c

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

int isEmpty(ListNodePtr sPtr){return sPtr == NULL;}

int isFull(ListNodePtr sPtr){return 0;}

void push(ListNodePtr *sPtr, char value){
  ListNodePtr newPtr;
... See lecture notes 5}

char pop(ListNodePtr *sPtr){
  ListNodePtr currPtr;
  char value;
... See lecture notes 5}

void printList (ListNodePtr currPtr){
... See lecture notes 4}
```

stack.h

```
struct listNode
{
  char data;
  struct listNode *nextPtr;
};

typedef struct listNode ListNode;
typedef ListNode *ListNodePtr;

int isEmpty(ListNodePtr sPtr);

int isFull(ListNodePtr sPtr);

void push(ListNodePtr *sPtr, char value);

char pop(ListNodePtr *sPtr);

void printList (ListNodePtr currPtr);
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

main(){
  ListNodePtr startPtr = NULL;
  char item;
  int noOfNodes = 0;
  printf("Write data: ");
  scanf("\n%c",&item);
  while (item != 'q')    {
      push(&startPtr, item);
      printf("Write data: ");
      scanf("\n%c",&item);
      noOfNodes ++;    }
  printf("%d\n", noOfNodes);
  printf("The contents of the stack: \n");
  printList(startPtr);}

% gcc main.c stack.c -o runme
```

Appendix: Encouraging good coding

Techniques for achieving good coding practices (in a project)

- Assign two people to every part of the project
- Review every line of code
- Require code sign-offs
- Emphasize that code listings are public assets
- Reward good code

(Code Complete: A practical handbook of software construction, Steve McConnell)