: http://www.netams.com/cvs/cvsweb.cgi/~checkout~/netams/addon/subnet-sum.c?rev=1.6&content-type=text/plain
NeTAMS 3.4

```
/* This programs eats strings with network data from stdin in format
*  x.x.x.x - y.y.y.y or x.x.x.x/bits
*  summarize it and produce output in format
*  suitble for netams target file and some other programms
*
*  (c) 2005 Yuriy N. Shkandybin
*  $Id: subnet-sum.c,v 1.6 2009-01-31 14:02:16 anton Exp $


STEP 0: build this source with:
        g++ -o subnet-sum subnet-sum.c

STEP1: download the latest RIPE database
        wget ftp://ftp.ripe.net/ripe/dbase/split/ripe.db.inetnum.gz

STEP2: unpack
        gunzip ripe.db.inetnum.gz

STEP3: create a script (vi ripe-networks-get.pl):
        #!/usr/bin/perl -w
        my $country="RU"; # replace for "UA" if needed
        while ( <STDIN> ) {
        if (substr($_, 0, 8) eq "inetnum:") {
         @a=split(/\s+/, $_); $inetnum=$a[1]." - ".$a[3]; }
        if (substr($_, 0, 8) eq "country:") {
         @a=split(/\s+/, $_);  if ($a[1] eq $country) {  print $inetnum."\n";  } }
        }

STEP4: process the database:
        ./ripe-networks-get.pl < ripe.db.inetnum > RU-ripe.db.inetnum

STEP5: process the regional database:
        ./subnet-sum < RU-ripe.db.inetnum > RU-ripe.db.inetnum.txt

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

typedef struct node {
        u_char full;
        struct node *up;
        struct node *left;
        struct node *right;
} NODE;

NODE *root;

#define MLEN2MASK(mlen) ((mlen)?(((in_addr_t)~(uint32_t)0) << (32 - (mlen))):0)
#define ANY        0xFFFFFFFF

u_char addr2tree(in_addr_t addr, u_char mlen);
void freetree(NODE *ptr);
void showtree(NODE *ptr, in_addr_t addr, u_char depth);

int  main () {
        FILE *f;
        char buf[64];
        char *ptr;
```

```
        struct in_addr addr1, addr2, tmp;
        in_addr_t a1, a2, addr;
        u_char mask;

        f=stdin;
//        f=fopen("1","r");

        //initialize root for bi-tree
        root=(NODE*)malloc(sizeof(NODE));
        root->full=0;
        root->up=root->left=root->right=NULL;

        while(!feof(f) && fgets(buf, 63, f)) {

                if(*buf=='#') continue;

                ptr=strchr(buf, '-');
                if(ptr) {
                        *ptr++=0;
                        while(ptr && *ptr==' ') ptr++;

                        inet_aton(buf,&addr1); a1=htonl(addr1.s_addr);
                        inet_aton(ptr,&addr2); a2=htonl(addr2.s_addr);
                        for(addr=a1; addr <= a2; addr++) {
                                if(addr == (addr & 0xFFFFFF00) && a2>= addr+255) { //speed up things
                                        mask = addr2tree( addr, 24);
                                        addr+=255;
                                }
                                tmp.s_addr=ntohl(addr);
                                mask = addr2tree( addr, 32);
//                                printf("add %s/%u\n", inet_ntoa(tmp), mask);
//                                showtree(root, 0, 0);
//                                printf("----------\n");
                        }
                } else if((ptr=strchr(buf, '/'))) {
                        *ptr=0;
                        ptr++;
                        inet_aton(buf,&addr1);
                        mask=atoi(ptr);
                        tmp.s_addr=addr1.s_addr&ntohl(MLEN2MASK(mask));
                        mask = addr2tree( htonl(tmp.s_addr), mask);
//                        printf("add %s/%u\n", inet_ntoa(tmp), mask );
//                        showtree(root, 0, 0);
//                        printf("----------\n");
                }
        }
//        printf("----------------------\n");
        showtree(root, 0, 0);
        freetree(root);
        return 0;
}

u_char addr2tree(in_addr_t addr, u_char mlen){
        NODE *ptr, *tmp;
        in_addr_t num;
        u_char i;

        ptr=root;

        for (i=0;i<mlen;i++) {
                num = 1<<(31-i);
                if(addr&num) {
                        if(!ptr->right) {
                                tmp=(NODE*)malloc(sizeof(NODE));
                                ptr->right=tmp;
                                tmp->up=ptr;
                                tmp->left=tmp->right=NULL;
                             tmp->full=0;
                          }
                                ptr=ptr->right;
                } else {
```

```
                        if(!ptr->left) {
                                tmp=(NODE*)malloc(sizeof(NODE));
                                ptr->left=tmp;
                                tmp->up=ptr;
                                tmp->left=tmp->right=NULL;
                                tmp->full=0;
                        }
                        ptr=ptr->left;
                }
        }
        ptr->full=1;

        //summarize if possible
        for (; i>0; i--) {
                tmp=ptr->up;
                if(!(tmp->left && tmp->left->full &&
                   tmp->right && tmp->right->full)) {
                        break;
                }
                ptr=tmp;
                ptr->full=1;
        }

        if(ptr->left) {
                freetree(ptr->left);
                ptr->left=NULL;
        }
        if(ptr->right) {
                freetree(ptr->right);
                ptr->right=NULL;
        }
        return i;
}

void freetree(NODE *ptr) {
        if(ptr->left) freetree(ptr->left);
        if(ptr->right) freetree(ptr->right);
        free(ptr);
}

void showtree(NODE *ptr, in_addr_t addr, u_char depth) {
        if(ptr->full) {
                struct in_addr ia;
                ia.s_addr=ntohl(addr<<(32-depth));
                printf("%s/%u\n", inet_ntoa(ia), depth);
        } else {
                if(ptr->left)  showtree(ptr->left,addr<<1,depth+1);
                if(ptr->right) showtree(ptr->right,(addr<<1)|0x00000001,depth+1);
        }
}
```