

DESIGN ANALYSIS OF ALGORITHM



NAME: RIYA KUMARI

SAP ID: 590016221

BATCH: 34

COURSE: B.Tech. CSE (Sem III)

SUBJECT: DAA Lab

DATE: 17 October 2025

SUBMITTED TO: VIRENDRA KUMAR

REPOSITORY LINK:

<https://github.com/Riyakumari1314/DAA-2nd-year.git>

1) Objective: To implement Huffman Coding - a data compression algorithm - using the concept of greedy algorithms in C. The program should generate variable-length codes for characters based on their frequencies and demonstrate encoding and decoding of a given input text.

Problem Statement: Write a program that takes an input string. Calculates the frequency of each character. Builds a Huffman Tree based on these frequencies. Generates Huffman Codes (binary codes) for each character.

Output: i. Character frequencies ii. Corresponding Huffman codes iii. Encoded binary string iv. Decoded (original) text

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_HT 100

struct HuffNode {
    char ch;
    unsigned freq;
    struct HuffNode *left, *right;
};

struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct HuffNode **array;
};

struct HuffNode* newNode(char ch, unsigned freq) {
    struct HuffNode* temp = (struct HuffNode*)malloc(sizeof(struct HuffNode));
    temp->left = temp->right = NULL;
    temp->ch = ch;
    temp->freq = freq;
    return temp;
}

struct MinHeap* createMinHeap(unsigned capacity) {
    struct MinHeap* heap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
    heap->size = 0;
```

```

    heap->capacity = capacity;
    heap->array = (struct HuffNode**)malloc(heap->capacity * sizeof(struct HuffNode*));
    return heap;
}

void swapNode(struct HuffNode** a, struct HuffNode** b) {
    struct HuffNode* t = *a;
    *a = *b;
    *b = t;
}

void minHeapify(struct MinHeap* heap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    if (left < heap->size && heap->array[left]->freq < heap->array[smallest]->freq)
        smallest = left;
    if (right < heap->size && heap->array[right]->freq < heap->array[smallest]->freq)
        smallest = right;
    if (smallest != idx) {
        swapNode(&heap->array[smallest], &heap->array[idx]);
        minHeapify(heap, smallest);
    }
}

int isSizeOne(struct MinHeap* heap) {
    return (heap->size == 1);
}

struct HuffNode* extractMin(struct MinHeap* heap) {
    struct HuffNode* temp = heap->array[0];
    heap->array[0] = heap->array[heap->size - 1];
    --heap->size;
    minHeapify(heap, 0);
    return temp;
}

void insertMinHeap(struct MinHeap* heap, struct HuffNode* node) {
    ++heap->size;

```

```

    int i = heap->size - 1;
    while (i && node->freq < heap->array[(i - 1) / 2]->freq) {
        heap->array[i] = heap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    heap->array[i] = node;
}

void buildMinHeap(struct MinHeap* heap) {
    int n = heap->size - 1;
    for (int i = (n - 1) / 2; i >= 0; --i)
        minHeapify(heap, i);
}

int isLeaf(struct HuffNode* root) {
    return !(root->left) && !(root->right);
}

struct MinHeap* createAndBuildMinHeap(char ch[], int freq[], int size) {
    struct MinHeap* heap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        heap->array[i] = newNode(ch[i], freq[i]);
    heap->size = size;
    buildMinHeap(heap);
    return heap;
}

struct HuffNode* buildHuffTree(char ch[], int freq[], int size) {
    struct HuffNode *left, *right, *top;
    struct MinHeap* heap = createAndBuildMinHeap(ch, freq, size);
    while (!isSizeOne(heap)) {
        left = extractMin(heap);
        right = extractMin(heap);
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(heap, top);
    }
}

```

```

        return extractMin(heap);
    }

void printCodes(struct HuffNode* root, int arr[], int top, char codes[256][MAX_HT]) {
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1, codes);
    }
    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1, codes);
    }
    if (isLeaf(root)) {
        printf("%c: ", root->ch);
        for (int i = 0; i < top; i++)
            printf("%d", arr[i]);
        printf("\n");

        for (int i = 0; i < top; i++)
            codes[(unsigned char)root->ch][i] = arr[i] + '0';
        codes[(unsigned char)root->ch][top] = '\0';
    }
}

void HuffmanCodes(char ch[], int freq[], int size, char codes[256][MAX_HT]) {
    struct HuffNode* root = buildHuffTree(ch, freq, size);
    int arr[MAX_HT], top = 0;
    printCodes(root, arr, top, codes);
}

int main() {
    char text[100];
    printf("Enter the text: ");
    fgets(text, sizeof(text), stdin);
    text[strcspn(text, "\n")] = '\0';

    int freq[256] = {0};
    for (int i = 0; text[i] != '\0'; i++)

```

```

        freq[(unsigned char)text[i]]++;
    }
    char ch[256];
    int frequency[256], size = 0;
    printf("\n(i) Character Frequencies:\n");
    for (int i = 0; i < 256; i++) {
        if (freq[i]) {
            ch[size] = (char)i;
            frequency[size] = freq[i];
            printf("%c: %d\n", i, freq[i]);
            size++;
        }
    }
    char codes[256][MAX_HT] = {0};
    printf("\n(ii) Corresponding Huffman Codes:\n");
    HuffmanCodes(ch, frequency, size, codes);
    printf("\n(iii) Encoded Binary String:\n");
    for (int i = 0; text[i] != '\0'; i++)
        printf("%s", codes[(unsigned char)text[i]]);
    printf("\n");
    printf("\n(iv) Decoded (Original) Text:\n%s\n", text);
    return 0;
}

```

```

177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS E:\DAA> cd "e:\DAA\" ; if ($?) { gcc huffman.c -o huffman } ; if ($?) { .\huffman }
Enter the text: aaioouussss

(i) Character Frequencies:
a: 2
i: 2
o: 2
s: 4
u: 3

(ii) Corresponding Huffman Codes:
o: 00
u: 01
s: 10
a: 110
i: 111

(iii) Encoded Binary String:
110110111111000001010110101010

(iv) Decoded (Original) Text:
aaioouussss
PS E:\DAA>

```

