DBMS Practical

Abhay Shanker Pathak

2021-01-20

Contents

1 Program for Recursive Binary & Linear Search **2**

1.1 Code 2

1.2 output 3

2 Program to sort 5 numbers using Heap Sorting method **3**

2.1 Code 3

2.2 Output 4

3 Program to sort 5 numbers using Merge Sort **4**

3.1 Code 4

3.2 Output 5

4 Program for Selection Sort **6**

4.1 Code 6

4.2 Output 6

5 Program for Insertion Sort **7**

5.1 Code 7

5.2 Output 7

6 Program for Quick Sort **8**

6.1 Code 8

6.2 Output 9

7 Knapsack Problem using Greedy Solution **9**

7.1 Code 9

7.2 Output 10

8 Perform Travelling Salesman Problem **11**

8.1 Code 11

8.2 Output 12

9 Find minium spanning tree using Kruskal’s algorithm **13**

9.1 Code 13

9.2 Output 15

10 Implementation of N Queen Problem using Backtracking **15**

10.1Code 15

10.2Output 16

11 Compilation done with help of Makefile **17**

12 Tools used in creating this practical(pdf) **18**

1 Program for Recursive Binary & Linear Search

1.1 Code

```
#include <iostream>
#include <array>
#define SIZE 20

bool linear_search(const std::array<int, SIZE>& arr, const size_t size, int key);
bool binary_search(const std::array<int, SIZE>& arr, size_t start, size_t end, const int key);

int main(int argc, char **argv) {
    size_t size;
    std::cout << "Enter the elements count to enter: " << std::endl;
    std::cin >> size;
    std::cout << "Enter elements of array[max count: 20]" << std::endl;
    std::array<int, SIZE> myarr;
    size_t i = 0;
    while (i < size) {
        std::cin >> myarr[i++];
    }
    int key;
    std::cout << "Enter element to search: " << std::endl;
    std::cin >> key;
    std::cout << (linear_search(myarr, size, key) ? "True" : "False") << "\n";
    std::cout << (binary_search(myarr, 0, SIZE - 1, key) ? "True" : "False") << std::endl;
    return 0;
}

/** linear search */
bool linear_search(const std::array<int, SIZE>& arr, const size_t size, int key) {
    for (size_t i = 0; i < size; ++i) {
        if (arr[i] == key) {
            return true;
        }
    }
    return false;
}

/** recursive binary search */
bool binary_search(const std::array<int, SIZE>& arr, size_t start, size_t end, int key) {
    if (start <= end) {
        if (arr[start] == key) {
            return true;
        }
        else {
            size_t mid = (start + end) / 2;
            if (key == arr[mid]) { return true; }
            if (key < arr[mid]) {
                return binary_search(arr, 0, mid - 1, key);
            }
            else {
                return binary_search(arr, mid + 1, end, key);
            }
        }
    }
    return false;
}
```

1.2 output

```
$ ./linear_binary_search
Enter the elements count to enter:
3
Enter elements of array[max count: 20]
3
4
5
Enter element to search:
4
True
True
```

2 Program to sort 5 numbers using Heap Sorting method

2.1 Code

```
#include <iostream>
#include <array>
#define SIZE 5

void manage(std::array<int, SIZE>& arr, int i);
void heapsort(std::array<int, SIZE>& arr, int i, size_t size);

int main(int argc, char **argv) {
    std::array<int, SIZE> myarr;
    std::cout << "\n--- Heap Sorting ---\n\n";
    std::cout << "Enter the elements in array to sort: " << std::endl;
    for (size_t i = 0; i < SIZE; ++i) {
        std::cin >> myarr[i];
        manage(myarr, i);
    }
    size_t j = SIZE;
    for (size_t i = 1; i <= SIZE; ++i) {
        int temp = myarr[1];
        myarr[1] = myarr[j];
        myarr[j--] = temp;
        heapsort(myarr, 1, j);
    }
    std::cout << "\n---Sorted Elements---\n";
    for (int i = 0; i < SIZE; ++i) {
        std::cout << myarr[i] << "\n";
    }
    std::cout << std::endl;
    return 0;
}

void manage(std::array<int, SIZE>& arr, int i) {
    int tmp;
    tmp = arr[i];
    while ((i > 1) && (arr[i / 2] < tmp)) {
        arr[i] = arr[i/2];
        i /= 2;
    }
    arr[i] = tmp;
}

void heapsort(std::array<int, SIZE>& arr, int i, size_t size) {
    int tmp, j;
    tmp = arr[i];
    j = i * 2;
    while (j <= size) {
        if ((j < size) && (arr[j] < arr[j + 1])) {
            j++;
        }
        if ((arr[j] < arr[j / 2])) { break; }
        arr[j / 2] = arr[j];
        j *= 2;
    }
    arr[j/2] = tmp;
}
```

2.2 Output

```
$ ./heapsort

--- Heap Sorting ---

Enter the elements in array to sort:
3
8
1
9
7

---Sorted Elements---
1
3
7
8
9
```

3 Program to sort 5 numbers using Merge Sort

3.1 Code

```
#include <iostream>
#include <array>
#define SIZE 5

void part(std::array<int, SIZE>& arr, size_t min, size_t max);
void merge(std::array<int, SIZE>& arr, size_t min, size_t mid, size_t max);

int main(int argc, char **argv) {
    std::array<int, SIZE> myarr;
    std::cout << "Enter the numbers in array: " << std::endl;
    for (size_t i = 0; i < SIZE; ++i) {
        std::cin >> myarr[i];
    }
    part(myarr, 0, SIZE - 1);
    for (int num: myarr) {
        std::cout << num << "\t";
    }
    std::cout << std::endl;
    return 0;
}

void part(std::array<int, SIZE>& arr, size_t min, size_t max) {
    size_t mid;
    if (min < max) {
        mid = (min + max) / 2;
        part(arr, min, mid);
        part(arr, mid + 1, max);
        merge(arr, min, mid, max);
    }
}

void merge(std::array<int, SIZE>& arr, size_t min, size_t mid, size_t max) {
    std::array<int, SIZE> tmp_arr;
    size_t i{}, j = min, k{}, m = mid + 1;
    for (i = min; j <= mid && m <= max; ++i) {
        if (arr[j] <= arr[m]) {
            tmp_arr[i] = arr[j++];
        }
        else {
            tmp_arr[i] = arr[m++];
        }
    }
    if (j > mid) {
        for (k = m; k <= max; ++k) {
            tmp_arr[i++] = arr[k];
        }
    }
    else {
        for (k = j; k <= mid; ++k) {
            tmp_arr[i++] = arr[k];
        }
    }
    for (k = min; k <= max; ++k) {
        arr[k] = tmp_arr[k];
    }
}
```

3.2 Output

```
$ ./merge_sorting
Enter the numbers in array:
4
3
2
```

```
0
1
0       1       2       3       4
```

4 Program for Selection Sort

4.1 Code

```cpp
#include <iostream>
#include <array>
#define SIZE 10

void selection_sort(std::array<int, SIZE>& arr);

int main(int argc, char **argv) {
    std::array<int, SIZE> myarr;
    std::cout << "Enter 10 integers in array: " << std::endl;
    for (int& num: myarr) {
        std::cin >> num;
    }
    std::cout << "Array after sorting\n\n";
    selection_sort(myarr);
    for (const int& num: myarr) {
        std::cout << num << "\t";
    }
    std::cout << std::endl;
    return 0;
}

void selection_sort(std::array<int, SIZE>& arr) {
    size_t pos{};
    int swap{};
    for (size_t c = 0; c < (SIZE - 1); ++c) {
        pos = c;
        for (size_t d = c + 1; d < SIZE; ++d) {
            if (arr[pos] > arr[d]) {
                pos = d;
            }
        }
        if (pos != c) {
            swap = arr[c];
            arr[c] = arr[pos];
            arr[pos] = swap;
        }
    }
}
```

4.2 Output

```
Enter 10 integers in array:
5
6
2
3
8
```

```
9
0
1
4
7
Array after sorting

0       1       2       3       4       5       6       7       8       9
```

5 Program for Insertion Sort

5.1 Code

```
#include <iostream>
#include <array>
#define SIZE 5

typedef std::array<int, SIZE> tarr;

void insertion_sort(tarr& arr);

int main(int argc, char **argv) {
    tarr myarr;
    std::cout << "Enter 5 elements:" << std::endl;
    for (int& num: myarr) {
        std::cin >> num;
    }
    insertion_sort(myarr);
    std::cout << "array after sorting: \n\n";
    for (const int& num: myarr) {
        std::cout << num << "\t";
    }
    std::cout << std::endl;
    return 0;
}

void insertion_sort(tarr& arr) {
    for (size_t c = 1; c <= SIZE - 1; ++c) {
        size_t d = c;
        while (d > 0 && arr[d] < arr[d - 1]) {
            int temp = arr[d];
            arr[d] = arr[d - 1];
            arr[d - 1] = temp;
            d--;
        }
    }
}
```

5.2 Output

```
$ ./insertion_sorting
Enter 5 elements:
5
8
9
```

```
2
1
array after sorting:

1       2       5       8       9
```

6 Program for Quick Sort

6.1 Code

```cpp
#include <iostream>
#include <array>
#define SIZE 5

typedef std::array<int, SIZE> tarr;

void quicksort(tarr& arr, size_t begin, size_t end);

int main(int argc, char **argv) {
    tarr myarr;
    std::cout << "Enter 5 elements for sorting: " << std::endl;
    for (int& num: myarr) {
        std::cin >> num;
    }
    quicksort(myarr, 0, SIZE - 1);
    std::cout << "array after sorting:\n\n" << std::endl;
    for (const int& num: myarr) {
        std::cout << num << "\t";
    }
    std::cout << std::endl;
    return 0;
}

void quicksort(tarr& arr, size_t begin, size_t end) {
    size_t pivot{}, i{}, j{};
    int temp{};
    if (begin < end) {
        pivot = begin;
        i = begin;
        j = end;

        while (i < j) {
            while (arr[i] <= arr[pivot] && i < end) { i++; }
            while (arr[j] > arr[pivot]) { j--; }
            if (i < j) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        temp = arr[pivot];
        arr[pivot] = arr[j];
        arr[j] = temp;
        quicksort(arr, begin, j - 1);
        quicksort(arr, j + 1, end);
    }
}
```

6.2 Output

```
Enter 5 elements for sorting:
6
9
2
4
7
array after sorting:

2       4       6       7       9
```

7 Knapsack Problem using Greedy Solution

7.1 Code

```
#include <iostream>
#include <array>

#define SIZE 10

struct Knapsack_lists {
    std::array<int, SIZE> profits;
    std::array<size_t, SIZE> weights;
    std::array<double, SIZE> profit_weights_ratio;
};

double get_profit_knapsack(const Knapsack_lists& sack_items, size_t& capacity);
void sort_according_to_ratio(Knapsack_lists& sack_items, size_t index);
size_t get_max_index(const std::array<double, SIZE>& pw_ratio, size_t index, size_t min_index);

int main(int argc, char **argv) {
    Knapsack_lists sack_items;
    sack_items.profits = {8, 56, 53, 74, 51, 94, 72, 14, 64, 12};
    sack_items.weights = {24, 44, 27, 54, 37, 67, 39, 28, 47, 25};
    for (size_t i = 0; i < SIZE; ++i) {
        sack_items.profit_weights_ratio[i] = sack_items.profits[i] / (double)sack_items.weights[i];
    }
    std::cout << "Enter the knapsack capacity: " << std::endl;
    size_t sack_capacity;
    std::cin >> sack_capacity;
    sort_according_to_ratio(sack_items, 0);
    double profit = get_profit_knapsack(sack_items, sack_capacity);
    std::cout << "Profit is: " << profit << std::endl;
    return 0;
}

void sort_according_to_ratio(Knapsack_lists& sack_items, size_t index) {
    if (index != SIZE - 1) {
        size_t get_next_min_index = get_max_index(sack_items.profit_weights_ratio, index + 1, index + 1);
        if ((index != get_next_min_index) &&
                (sack_items.profit_weights_ratio[index] < sack_items.profit_weights_ratio[get_next_min_inde
            double temp_ratio = sack_items.profit_weights_ratio[index];
            sack_items.profit_weights_ratio[index] = sack_items.profit_weights_ratio[get_next_min_index];
            sack_items.profit_weights_ratio[get_next_min_index] = temp_ratio;

            int temp_profit = sack_items.profits[index];
            sack_items.profits[index] = sack_items.profits[get_next_min_index];
            sack_items.profits[get_next_min_index] = temp_profit;

            double temp_weight = sack_items.weights[index];
            sack_items.weights[index] = sack_items.weights[get_next_min_index];
            sack_items.weights[get_next_min_index] = temp_weight;
        }
        sort_according_to_ratio(sack_items, index + 1);
    }
}

size_t get_max_index(const std::array<double, SIZE>& pw_ratio, size_t index, size_t min_index) {
    if (index == SIZE) {
        return min_index;
    }
    if (pw_ratio[index] > pw_ratio[min_index]) {
        min_index = index;
    }
    return get_max_index(pw_ratio, index + 1, min_index);
}

double get_profit_knapsack(const Knapsack_lists& sack_items, size_t& capacity) {
    size_t count = 0;
    double profit = 0;
    while (capacity != 0) {
        if (capacity < sack_items.weights[count]) {
            double partition = capacity / (double)sack_items.weights[count];
            capacity = 0;
            profit += (sack_items.profits[count] * partition);
        }
        else {
            capacity -= sack_items.weights[count];
            profit += sack_items.profits[count];
        }
        count++;
    }
    return profit;
}
```

7.2 Output

```
$ ./knapsack_greedy
Enter the knapsack capacity:
45
Profit is: 86.2308

$ ./knapsack_greedy
Enter the knapsack capacity:
76
```

```
Profit is: 139.03
```

8 Perform Travelling Salesman Problem

8.1 Code

```
#include <iostream>
#include <array>
#define SIZE 5

typedef std::array<std::array<int, SIZE>, SIZE> tmatrix;
typedef std::array<int, SIZE> tarr;

void minimum_cost(tmatrix& matrix, size_t city, tarr& visited_cities, int& cost, size_t& city_limit);
void minimum_cost(tmatrix& matrix, size_t city, tarr& visited_cities, int& cost, size_t& city_limit);

int main(int argc, char **argv) {
    size_t city_limit{};
    int cost{};
    std::cout << "Enter total number of cities: " << std::endl;
    std::cin >> city_limit;
    tmatrix matrix;
    std::cout << "Enter Cost Matrix" << std::endl;
    for (tarr& row: matrix) {
        for (int& num: row) {
            std::cin >> num;
        }
    }
    tarr visited_cities;  // initialized all blocks to 0
    std::cout << "\nEntered Cost Matrix is:\n";
    for (const tarr& row: matrix) {
        for (const int& num: row) {
            std::cout << num << "\t";
        }
        std::cout << "\n";
    }
    std::cout << "\nPath: " << std::endl;
    minimum_cost(matrix, 0, visited_cities, cost, city_limit);
    std::cout << "\nMinimum cost: " << cost << std::endl;
    return 0;
}

void minimum_cost(tmatrix& matrix, size_t city, tarr& visited_cities, int& cost, size_t& city_limit) {
    size_t nearest_city{};
    visited_cities[city] = 1;
    std::cout << city + 1 << " " << std::endl;
    nearest_city = tsp(matrix, city, city_limit, visited_cities, cost);
    if (nearest_city == 999) {
        nearest_city = 0;
        std::cout << nearest_city + 1;
        cost += matrix[city][nearest_city];
        return;
    }
    minimum_cost(matrix, nearest_city, visited_cities, cost, city_limit);
}
```

```
int tsp(tmatrix& matrix, int c, size_t& city_limit, tarr& visited_cities, int& cost) {
    size_t nearest_city = 999, minimum = 999;
    int temp{};
    for (size_t count = 0; count <= city_limit; ++count) {
        if ((matrix[c][count] != 0) && (visited_cities[count] == 0)) {
            if (matrix[c][count] < minimum) {
                minimum = matrix[count][0] + matrix[c][count];
            }
            temp = matrix[c][count];
            nearest_city = count;
        }
    }
    if (minimum != 999) {
        cost += temp;
    }
    return nearest_city;
}
```

8.2 Output

```
1 $ ./travelling_salesman
Enter total number of cities:
4
Enter Cost Matrix
2
1
5
4
2
4
6
4
8
2
1
5
6
4
7
8
3
4
5
2
4
6
8
4
9

Entered Cost Matrix is:
2       1       5       4       2
4       6       4       8       2
1       5       6       4       7
8       3       4       5       2
```

```
4       6       8       4       9

Path:
1
4
1
Minimum cost: 12
```

9 Find minium spanning tree using Kruskal’s algorithm

9.1 Code

```cpp
#include <iostream>
#include <array>
#define MAX 5

typedef std::array<std::array<int, MAX>, MAX> tarr2d;
typedef std::array<int, MAX> tarr;

typedef struct {
    int u, v, w;
} edge;

typedef struct {
    edge data[MAX];
    int n;
} edgelist;

edgelist elist; // declaring and initializing edgelist

tarr2d G;
size_t n{};

edgelist spanlist;

void apply_kruskal();
void print_result();
void sort();
int find(const tarr& belongs, int vertexno);
void union1(tarr& belongs, int c1, int c2);

int main(int argc, char **argv) {
    //size_t num_vertices;
    //std::cout << "Enter number of vertices: " << std::endl;
    //std::cin >> num_vertices;
    std::cout << "Enter adjecency matrix: " << std::endl;
    for (tarr& row: G) {
        for (int& num: row) {
            std::cin >> num;
        }
    }
    apply_kruskal();
    print_result();
    return 0;
}
```

```
void apply_kruskal() {
    tarr belongs;
    int cno1{}, cno2{};
    elist.n = 0;

    for (size_t i = 1; i < n; ++i) {
        for (size_t j = 0; j < i; ++j) {
            if (G[i][j] != 0) {
                elist.data[elist.n].u = i;
                elist.data[elist.n].v = j;
                elist.data[elist.n].w = G[i][j];
                elist.n++;
            }
        }
    }
    sort();

    for (size_t i = 0; i < n; ++i) {
        belongs[i] = i;
    }
    spanlist.n = 0;

    for (size_t i = 0; i < n; ++i) {
        cno1 = find(belongs, elist.data[i].u);
        cno2 = find(belongs, elist.data[i].v);
        if (cno1 != cno2) {
            spanlist.data[spanlist.n] = elist.data[i];
            spanlist.n++;
            union1(belongs, cno1, cno2);
        }
    }
}

int find(const tarr& belongs, int vertexno) {
    return belongs[vertexno];
}

void union1(tarr& belongs, int c1, int c2) {
    for (size_t i = 0; i < n; ++i) {
        if (belongs[i] == c2) {
            belongs[i] = c1;
        }
    }
}

void sort() {
    edge temp;

    for (size_t i = 1; i < elist.n; ++i) {
        for (size_t j = 0; j < elist.n - 1; ++j) {
            if (elist.data[j].w > elist.data[j + 1].w) {
                temp = elist.data[j];
                elist.data[j] = elist.data[j + 1];
                elist.data[j + 1] = temp;
            }
        }
    }
}

void print_result() {
    int cost = 0;
    for (size_t i = 0; i < spanlist.n; ++i) {
        std::cout << spanlist.data[i].u << "\t"
            << spanlist.data[i].v << "\t"
            << spanlist.data[i].w << "\n";
        cost += spanlist.data[i].w;
    }
    std::cout << "\nCost of the spanning tree: " << cost << std::endl;
}
```

9.2 Output

No output, seg fault

10 Implementation of N Queen Problem using Backtracking

10.1 Code

```
#include <iostream>
#include <cmath>
#include <array>
#define MAX 5

typedef std::array<std::array<char, MAX>, MAX> tarr2d;
typedef std::array<char, MAX> tarr;

tarr2d a;

void nqueen(int row, const size_t& nq);
int feasible(int row, size_t col, const size_t& nq);
int getmarkedcol(size_t row, const size_t& nq);
void printmatrix();

int main(int argc, char **argv) {
    size_t num_queens;
    std::cout << "Enter number of queens: " << std::endl;
    std::cin >> num_queens;
    for (size_t i = 0; i < num_queens; ++i) {
        for (size_t j = 0; j < num_queens; ++j) {
            a[i][j] = '.';
        }
    }
    nqueen(0, num_queens);
    return 0;
}

void nqueen(int row, const size_t& nq) {
    if (row < nq) {
        for (size_t i = 0; i < nq; ++i) {
            if (feasible(row, i, nq)) {
                a[row][i] = 'Q';
                nqueen(row + 1, nq);
                a[row][i] = '.';
            }
        }
    }
    else {
        std::cout << "\nThe solution is: \n";
        printmatrix();
    }
}

int feasible(int row, size_t col, const size_t& nq) {
    int tcol;
    for (size_t i = 0; i < nq; ++i) {
        tcol = getmarkedcol(i, nq);
        if (col == tcol || row - i == (col - tcol)) {
            return 0;
        }
    }
    return 1;
}

int getmarkedcol(size_t row, const size_t& nq) {
    for (size_t i = 0; i < nq; ++i) {
        if (a[row][i] == 'Q') {
            return i;
            break;
        }
    }
    return 0;
}

void printmatrix() {
    std::cout << "\n";
    for (const tarr& row: a) {
        for (const char& c: row) {
            std::cout << c << "\t";
        }
        std::cout << "\n";
    }
    std::cout << std::endl;
}
```

10.2 Output

```
$ ./nqueen_problem
Enter number of queens:
4

The solution is:
.   Q   .   .
.   .   .   Q
Q   .   .   .
.   .   Q   .
```

11 Compilation done with help of Makefile

Here’s is the makefile which I wrote to compile the programs:

```
# /* --- Makefile --- */

CC     = clang++
CFLAG  = -Wall -std=c++14
CDFLAG = -Wall -std=c++14 -g
LD     = clang++
LDFLAG = -v
LFLAG  =


SRC_DIR   = src
OBJ_DIR   = obj
INC_DIR   = inc
BIN_DIR   = bin
DEBUG_DIR = debug
DIRS      = ${BIN_DIR} ${OBJ_DIR} ${DEBUG_DIR}


SRC       = $(wildcard ${SRC_DIR}/*.cpp)
OBJ       = $(addprefix ${OBJ_DIR}/, $(notdir ${SRC:.cpp=.o}))
BIN       = ${BIN_DIR}/$(notdir $(realpath .))
DEBUG_OBJ = $(addprefix ${DEBUG_DIR}/, $(notdir ${SRC:.cpp=.o}))
DEBUG_BIN = $(addprefix ${DEBUG_DIR}/, $(notdir $(realpath .)))


all: dir ${BIN}

dir:
    mkdir -p ${DIRS}


${OBJ_DIR}/%.o: ${SRC_DIR}/%.cpp
    -@echo "compiling $? -> $@"
    ${CC} ${CFLAG} -I ${INC_DIR} -c -o $@ $^

${BIN}: ${OBJ}
    -@echo "Linking $? -> $@"
    ${LD} ${LFLAG} -o $@ ${OBJ_DIR}/*.o
    -@echo "copied ${BIN} -> $(notdir $(realpath .))"
    cp -f ${BIN} .


debug: dir ${DEBUG_BIN}

${DEBUG_DIR}/%.o: ${SRC_DIR}/%.cpp
    -@echo "compiling $? -> $@"
    ${CC} ${CDFLAG} -I ${INC_DIR} -c -o $@ $^

${DEBUG_BIN}: ${DEBUG_OBJ}
    -@echo "Linking to -> $@"
    ${LD} ${LDFLAG} -o $@ ${DEBUG_DIR}/*.o
```

```
clean:
    rm -rf ${DIRS} $(notdir $(realpath .))

.SILENT:
.PHONY: all dir debug clean
```

12 Tools used in creating this practical(pdf)

- **OS :** 5.4.85-1-MANJARO
- **WM** : DWM
- **Pdf(markup) convertor:** Pandoc(2.11.2)
- **Pdf engine :** xelatex
- **Source File Format :** Markdown(md)
- **Text Editor :** Neovim-nightly(v0.5.0-dev+1000-g84d08358b)
- **Compiler:** Clang(version 11.0.0)
- **Compiling uitlity:** GNU make(4.3)

--* THE END --*