# Contents

[Contents 1](#_Toc21168723)

[1. Sort 1](#_Toc21168724)

[1.1. Selection Sort 1](#_Toc21168725)

[1.2. Insertion Sort 2](#_Toc21168726)

[1.3. Bubble Sort 2](#_Toc21168727)

[1.4. Quick Sort 3](#_Toc21168728)

[1.5. Topological Sort 3](#_Toc21168729)

[2. Search 5](#_Toc21168730)

[2.1. Binary Search 5](#_Toc21168731)

[3. DP 6](#_Toc21168732)

[3.1. Knapsack 6](#_Toc21168733)

[3.2. LCS (longest common subsequence) 6](#_Toc21168734)

[3.3. LIS (longest increasing subsequence) 7](#_Toc21168735)

[3.4. Edit Distance 7](#_Toc21168736)

[3.5. MCM 8](#_Toc21168737)

[4. Shortest Path 8](#_Toc21168738)

[4.1. Dijkstra 8](#_Toc21168739)

[4.2. Bellman-Ford 10](#_Toc21168740)

[4.3. Floyd-Warshall 11](#_Toc21168741)

[5. String 12](#_Toc21168742)

[6. Permutation, Combination 13](#_Toc21168743)

[7. List 13](#_Toc21168744)

[8. Queue 14](#_Toc21168745)

[9. Stack 14](#_Toc21168746)

[10. Graph 15](#_Toc21168747)

[11. Tree 15](#_Toc21168748)

[12. Heap 15](#_Toc21168749)

# **Sort**

## **Selection Sort**

1. #include <stdio.h>
2. #define MAX 5005
3. int arr[MAX];
4. int N;
5. void Swap(int \*a, int \*b) {
6. int temp = \*a;
7. \*a = \*b;
8. \*b = temp;
9. }
10. void selection\_sort(int \*arr, int size) {
11. for (int i = 0; i < size; i++) {
12. int minidx = i;
13. for (int j = i + 1; j < size; j++) {
14. if (arr[minidx] > arr[j]) {
15. minidx = j;
16. }
17. }
18. Swap(&arr[minidx], &arr[i]);
19. }
20. }
21. int main() {
22. scanf("%d",&N);
23. for(int i=0; i<N; i++) {
24. scanf("%d",&arr[i]);
25. }
26. selection\_sort(arr, N);
27. for (int i = 0; i < N; i++) {
28. printf("%d ", arr[i]);
29. }
30. return 0;
31. }

## Insertion Sort

1. #include <stdio.h>
2. #define MAX 5005
3. int d[MAX], n;
4. int main() {
5. scanf("%d", &n);
6. for (int i = 0; i < n; i++)
7. scanf("%d", &d[i]);
8. for (int i = 0; i < n; i++) {
9. int temp = d[i];
10. int j = i - 1;
11. for (; j >= 0; j--) {
12. if (d[j] < temp)
13. break;
14. d[j + 1] = d[j];
15. }
16. d[j + 1] = temp;
17. }
18. for (int i = 0; i < n; i++)
19. printf("%d ", d[i]);
20. return 0;
21. }

## Bubble Sort

1. #include <stdio.h>
2. #define MAX 5005
3. int d[MAX];
4. void swap(int \*a, int \*b) {
5. int t = \*a;
6. \*a = \*b;
7. \*b = t;
8. }
9. int main()
10. {
11. int n;
12. scanf("%d", &n);
13. for(int i = 0 ; i < n ; i++)
14. scanf("%d", &d[i]);
15. for(int i = 0 ; i < n - 1 ; i++)
16. {
17. for(int j = 0; j < n - i - 1; j++)
18. {
19. if(d[j] > d[j + 1])
20. {
21. swap(&d[j], &d[j + 1]);
22. }
23. }
24. }
25. for(int i = 0 ; i < n ; i++)
26. printf("%d ", d[i]);
27. return 0;
28. }

## Quick Sort

1. #include <stdio.h>
2. void quickSort(int \*arr, int left, int right) {
3. int pivot, left\_temp, right\_temp;
4. left\_temp = left;
5. right\_temp = right;
6. pivot = arr[left];
7. while (left < right) {
8. while (arr[right] >= pivot && (left < right)) {
9. right--;
10. }
11. if (left != right) {
12. arr[left] = arr[right];
13. }
14. while (arr[left] <= pivot && (left < right)) {
15. left++;
16. }
17. if (left != right) {
18. arr[right] = arr[left];
19. right--;
20. }
21. }
22. arr[left] = pivot;
23. pivot = left;
24. left = left\_temp;
25. right = right\_temp;
26. if (left < pivot) quickSort(arr, left, pivot - 1);
27. if (right > pivot) quickSort(arr, pivot + 1, right);
28. }
29. int N;
30. int arr[100010];
31. int main() {
32. scanf("%d", &N);
33. for (int i = 0; i < N; i++) {
34. scanf("%d", &arr[i]);
35. }
36. quickSort(arr, 0, N - 1);
37. for (int i = 0; i < N; i++) {
38. printf("%d ", arr[i]);
39. }
40. }

## Topological Sort

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <assert.h>
4. typedef char bool;
5. #define MAX 55
6. typedef struct node {
7. struct node\* next;
8. int value;
9. } node;
10. void add(node\*\* head, int value) {
11. node\* new\_node = (node \*)malloc(sizeof(node));
12. new\_node->value = value;
13. if (!(\*head)) {
14. new\_node->next = NULL;
15. (\*head) = new\_node;
16. } else {
17. new\_node->next = (\*head);
18. (\*head) = new\_node;
19. }
20. return;
21. }
22. struct queue {
23. int \*data;
24. int front, rear;
25. int maxsize;
26. int size;
27. };
28. void queue\_init(struct queue\* q, int sz) {
29. q->maxsize = sz;
30. q->front = 0;
31. q->rear = 0;
32. q->size = 0;
33. q->data = (int\*)malloc(q->maxsize \* sizeof(int));
34. }
35. int queue\_size(struct queue\* q) {
36. return q->size;
37. }
38. bool queue\_empty(struct queue\* q) {
39. return queue\_size(q) == 0;
40. }
41. bool queue\_full(struct queue\* q) {
42. return q->size == q->maxsize;
43. }
44. void queue\_push(struct queue\* q, int val) {
45. assert(!queue\_full(q));
46. q->data[q->rear] = val;
47. q->rear = (q->rear + 1) % q->maxsize;
48. q->size++;
49. }
50. void queue\_pop(struct queue\* q) {
51. assert(!queue\_empty(q));
52. q->front = (q->front + 1) % q->maxsize;
53. q->size--;
54. }
55. int queue\_front(struct queue\* q) {
56. assert(!queue\_empty(q));
57. return q->data[q->front];
58. }
59. node \*head[MAX];
60. int ind[MAX];
61. int N, M;
62. int u, v;
63. int main() {
64. scanf("%d %d", &N, &M);
65. for (int i = 0; i < M; i++) {
66. scanf("%d %d", &u, &v);
67. add(&head[u], v);
68. ind[v]++;
69. }
70. struct queue q;
71. queue\_init(&q, MAX);
72. for (int i = 1; i <= N; i++) {
73. if (ind[i] == 0) {
74. queue\_push(&q,i);
75. }
76. }
77. while (!queue\_empty(&q)) {
78. int here = queue\_front(&q);
79. queue\_pop(&q);
80. printf("%d ", here);
81. node\* current\_node = head[here];
82. for (; current\_node; current\_node=current\_node->next) {
83. int there = current\_node->value;
84. ind[there]--;
85. if (ind[there] == 0) {
86. queue\_push(&q,there);
87. }
88. }
89. }
90. return 0;
91. }

# Search

## Binary Search

#include <stdio.h>

#include <stdbool.h>

#define MAX 5005

int d[MAX], n;

void swap(int \*a, int \*b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

bool bsearch(int val) {

int l = 0, r = n - 1;

while (l <= r) {

int mid = (l + r) / 2;

if (val == d[mid]) return true;

else if (val > d[mid]) l = mid + 1;

else r = mid - 1;

}

return false;

}

int main() {

scanf("%d", &n);

for (int i = 0; i < n; i++)

scanf("%d", &d[i]);

// sorting

for (int i = 0; i < n; i++)

for (int j = i + 1; j < n; j++)

if (d[i] > d[j]) {

swap(&d[i], &d[j]);

}

int query;

scanf("%d", &query);

while (query--) {

int x; scanf("%d", &x);

if (bsearch(x)) printf("exist\n");

else printf("not exist\n");

}

}

# DP

## Knapsack

#include <stdio.h>

int d[101][10001], v[101], w[101];

int main() {

int n, k;

scanf("%d %d", &n, &k);

for(int i = 1; i <= n; i++) {

scanf("%d %d", &v[i], &w[i]);

}

for(int i = 1; i <= n; i++) {

for(int j = 0; j <= k; j++) {

if(j < w[i]) {

d[i][j] = d[i - 1][j];

} else if(d[i - 1][j - w[i]] + v[i] > d[i - 1][j]) {

d[i][j] = d[i - 1][j - w[i]] + v[i];

} else {

d[i][j] = d[i - 1][j];

}

}

}

int ans = 0;

for(int i = 0; i <= k; i++){

if(ans < d[n][i]){

ans = d[n][i];

}

}

printf("%d\n", ans);

return 0;

}

## LCS (longest common subsequence)

#include <stdio.h>

int lcs[1001][1001];

char a[1001], b[1001];

int main() {

int la, lb;

scanf("%d %d", &la, &lb);

scanf("%s %s", a + 1, b + 1);

for(int i = 1; i <= la; i++) {

for(int j = 1; j <= lb; j++) {

if(a[i] == b[j]) {

lcs[i][j] = lcs[i - 1][j - 1] + 1;

} else {

if(lcs[i - 1][j] < lcs[i][j - 1]) {

lcs[i][j] = lcs[i][j - 1];

} else {

lcs[i][j] = lcs[i - 1][j];

}

}

}

}

printf("%d\n", lcs[la][lb]);

return 0;

}

## LIS (longest increasing subsequence)

#include <stdio.h>

int a[1000], d[1000];

int main() {

int n;

scanf("%d", &n);

for(int i = 0; i < n; i++) {

scanf("%d", &a[i]);

}

for(int i = 0; i < n; i++) {

for(int j = 0; j < i; j++) {

if(a[j] < a[i]) {

if(d[i] < d[j]) {

d[i] = d[j];

}

}

}

d[i]++;

}

int ans = 0;

for(int i = 0; i < n; i++) {

if(ans < d[i]) {

ans = d[i];

}

}

printf("%d\n", ans);

return 0;

}

## Edit Distance

#include <stdio.h>

int d[1001][1001];

char a[1001], b[1001];

int min(int a, int b) {

return a < b ? a : b;

}

int main() {

int la, lb;

scanf("%d %d", &la, &lb);

scanf("%s %s", a + 1, b + 1);

for(int i = 1; i <= la; i++) {

d[i][0] = i;

}

for(int j = 1; j <= lb; j++) {

d[0][j] = j;

}

for(int i = 1; i <= la; i++) {

for(int j = 1; j <= lb; j++) {

if(a[i] == b[j]) {

d[i][j] = d[i - 1][j - 1];

} else {

d[i][j] = min(min(d[i - 1][j], d[i][j - 1]), d[i - 1][j - 1]) + 1;

}

}

}

printf("%d\n", d[la][lb]);

return 0;

}

## MCM

#include <stdio.h>

int d[501][501], r[501], c[501];

int main() {

int n;

scanf("%d", &n);

for(int i = 0; i < n; i++) {

scanf("%d %d", &r[i], &c[i]);

}

for(int l = 1; l < n; l++) {

for(int i = 0; i < n - l; i++) {

int j = i + l;

d[i][j] = -1;

for(int k = i; k < j; k++) {

int calc = d[i][k] + d[k + 1][j] + r[i] \* c[k] \* c[j];

if(d[i][j] < 0) {

d[i][j] = calc;

} else if(calc < d[i][j]) {

d[i][j] = calc;

}

}

}

}

printf("%d\n", d[0][n - 1]);

return 0;

}

# Shortest Path

## Dijkstra

#include <stdio.h>

#include <stdlib.h>

#define MAX 55

typedef struct node {

struct node\* next;

int value;

int weight;

} node;

struct priority\_queue {

int x, cost;

} pq[MAX\*2], tmp;

int pq\_t;

void add(node\*\* head, int value, int weight);

void pq\_pop();

void pq\_push(int x, int cost);

int N, M, u, v, d, S;

node \*head[MAX];

int\* Dijkstra(int start) {

int \*dist = (int\*)malloc(sizeof(int)\*(N+1));

for (int i = 1; i <= N; i++) {

dist[i] = -1;

}

// first : dist , second : vertex\_pos

dist[start] = 0;

pq\_push(start, dist[start]);

while (pq[1].x!=0) {

int here = pq[1].x;

int heredist = pq[1].cost;

pq\_pop();

if (heredist > dist[here]) continue;

node \*current\_node = head[here];

for (; current\_node; current\_node=current\_node->next) {

int there = current\_node->value;

int nextdist = heredist + current\_node->weight;

if (dist[there] == -1 || dist[there] > nextdist) { //최단 거리 갱신

dist[there] = nextdist;

pq\_push(there, nextdist);

}

}

}

return dist;

}

int main() {

scanf("%d %d %d", &N, &M, &S);

for (int i = 0; i < M; i++) {

scanf("%d %d %d", &u, &v, &d);

add(&head[u], v, d);

}

int \*dist = Dijkstra(S);

for (int i = 1; i <= N; i++) {

printf("%d\n", dist[i]);

}

return 0;

}

void add(node\*\* head, int value, int weight) {

node\* new\_node = (node \*)malloc(sizeof(node));

new\_node->value = value;

new\_node->weight = weight;

if (!(\*head)) {

new\_node->next = NULL;

(\*head) = new\_node;

} else {

new\_node->next = (\*head);

(\*head) = new\_node;

}

return;

}

void pq\_pop() {

pq[1] = pq[pq\_t--];

for (int s = 1;;) {

int e = s \* 2;

if (e > pq\_t) return;

if (e + 1 <= pq\_t && pq[e].cost > pq[e + 1].cost) e++;

if (pq[s].cost > pq[e].cost) {

tmp = pq[s];

pq[s] = pq[e];

pq[e] = tmp;

s = e;

} else break;

}

}

void pq\_push(int x, int cost) {

pq[++pq\_t].cost = cost;

pq[pq\_t].x = x;

for (int i = pq\_t; i > 1; i /= 2) {

if (cost < pq[i / 2].cost) pq[i] = pq[i / 2];

else {

pq[i].x = x;

pq[i].cost = cost;

return;

}

}

pq[1].x = x;

pq[1].cost = cost;

}

## Bellman-Ford

#include <stdio.h>

#include <stdlib.h>

#define MAX 55

#define INF 987654321

typedef struct node {

struct node\* next;

int value;

int weight;

} node;

void add(node\*\* head, int value, int weight);

int N, M, u, v, d, S;

node \*head[MAX];

int\* bellmanFord(int start) {

int \*upper = (int\*)malloc(sizeof(int)\*(N + 1));

int updated = 0;

node \*current\_node;

int here,there;

int heredist,nextcost;

for (int i = 1; i <= N; i++) {

upper[i] = INF;

}

// first : dist , second : vertex\_pos

upper[start] = 0;

for (int iter = 0; iter < N; iter++) {

updated = 0;

for (int here = 1; here <= N; here++) {

current\_node = head[here];

for (; current\_node; current\_node=current\_node->next) {

int there = current\_node->value;

int nextcost = upper[here] + current\_node->weight;

if (upper[there] > nextcost) {

upper[there] = nextcost;

updated = 1;

}

}

}

if (!updated) break;

}

if (updated) return NULL;

return upper;

}

int main() {

scanf("%d %d %d", &N, &M, &S);

for (int i = 0; i < M; i++) {

scanf("%d %d %d", &u, &v, &d);

add(&head[u], v, d);

}

int \*dist = bellmanFord(S);

if (dist == NULL)

puts("-1");

else {

for (int i = 1; i <= N; i++) {

printf("%d\n", dist[i]);

}

}

return 0;

}

void add(node\*\* head, int value, int weight) {

node\* new\_node = (node \*)malloc(sizeof(node));

new\_node->value = value;

new\_node->weight = weight;

if (!(\*head)) {

new\_node->next = NULL;

(\*head) = new\_node;

} else {

new\_node->next = (\*head);

(\*head) = new\_node;

}

return;

}

## Floyd-Warshall

#include <stdio.h>

#define MAX 105

#define INT\_MAX 0x7fffffff

int n, d[MAX][MAX];

void floyd\_warwhall() {

for (int k = 1; k <= n; k++)

for (int i = 1; i <= n; i++)

for (int j = 1; j <= n; j++) {

if (d[i][k] == INT\_MAX || d[k][j] == INT\_MAX) continue;

if (d[i][j] > d[i][k] + d[k][j])

d[i][j] = d[i][k] + d[k][j];

}

}

int main() {

scanf("%d", &n);

for (int i = 1; i <= n; i++)

for (int j = 1; j <= n; j++) {

scanf("%d", &d[i][j]);

if (d[i][j] == 0) d[i][j] = INT\_MAX;

}

floyd\_warwhall();

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) printf("%d ", d[i][j]);

printf("\n");

}

}

# String

#include <iostream>

#include <string>

#include <algorithm>

#include <vector>

using namespace std;

int n, m;

string N, M;

vector<int> getPartialMatch(const string& M) {

int m = M.size();

vector<int> pi(m, 0);

int begin = 1, matched = 0;

while (begin + matched < m) {

if (M[begin + matched] == M[matched]) {

++matched;

pi[begin + matched - 1] = matched;

} else {

if (matched == 0)

++begin;

else {

begin += matched - pi[matched - 1];

matched = pi[matched - 1];

}

}

}

return pi;

}

vector<int> kmpSearch(const string & N, const string & M) {

vector<int> ret;

vector<int> pi = getPartialMatch(M);

int n = N.size(), m = M.size();

int begin = 0, matched = 0;

while (begin <= n - m) {

if (matched < m && N[begin + matched] == M[matched]) {

++matched;

if (matched == m) ret.push\_back(begin);

} else {

if (matched == 0) {

++begin;

} else {

begin += matched - pi[matched - 1];

matched = pi[matched - 1];

}

}

}

return ret;

}

int main() {

cin >> n >> N;

cin >> m >> M;

vector<int> sol= kmpSearch(N, M);

cout << '{';

for (int i = 0; i < sol.size()-1; i++) {

cout << sol[i] << ',';

}

if (!sol.empty())

cout << sol[sol.size() - 1];

cout << '}';

return 0;

}

# Permutation, Combination

#include <iostream>

using namespace std;

int comb[1001][1001];

void calc\_combination() {

comb[0][0] = 1;

for(int i = 1; i <= 1000; i++) {

for(int j = 0; j <= i; j++) {

if(j == 0) {

comb[i][j] = 1;

} else {

comb[i][j] = (comb[i - 1][j - 1] + comb[i - 1][j]) % 1000007;

}

}

}

}

int main() {

calc\_combination();

int q;

cin >> q;

while(q--) {

int n, r;

cin >> n >> r;

cout << comb[n][r] << "\n";

}

return 0;

}

# List

#include <stdio.h>

#include <list>

using namespace std;

int main() {

int n;

scanf("%d", &n);

list<int> mylist;

while(n--) {

char command[10];

scanf("%s", command);

// push, get, size, count, clear

if(command[0] == 'p') {

int value;

scanf("%d", &value);

mylist.push\_back(value);

} else if(command[0] == 'g') {

int idx;

scanf("%d", &idx);

list<int>::iterator it = mylist.begin();

while(idx--) {

it++;

}

printf("%d\n", \*it);

} else if(command[0] == 's') {

printf("%d\n", mylist.size());

} else if(command[1] == 'o') {

int target, cnt = 0;

scanf("%d", &target);

for(list<int>::iterator it = mylist.begin(); it != mylist.end(); it++) {

if(\*it == target) {

cnt++;

}

}

printf("%d\n", cnt);

} else {

mylist.clear();

}

}

return 0;

}

# Queue

#include <queue>

#include <iostream>

#include <string>

using namespace std;

int main() {

int val,N;

queue<int> q;

string cmd;

cin >> N;

for (int i = 0; i < N; i++) {

cin >> cmd;

if (cmd[0] == 's') {

cout << q.size() << endl;

} else if (cmd[0] == 'e') {

cin >> val;

q.push(val);

} else if (cmd[0] == 'd') {

q.pop();

} else if (cmd[0] == 'f') {

cout << q.front() << endl;

}

}

return 0;

}

# Stack

#include <stack>

#include <iostream>

#include <string>

using namespace std;

int N, val;

string cmd;

int main() {

stack<int> st;

cin >> N;

for (int i = 0; i < N; i++) {

cin >> cmd;

if (cmd[0] == 's') {

cout << st.size() << endl;

} else if (cmd[0] == 'p') {

if (cmd[1] == 'u') {

cin >> val;

st.push(val);

} else if (cmd[1] == 'o') {

st.pop();

}

} else if (cmd[0] == 't') {

cout << st.top() << endl;

}

}

return 0;

}

# Graph

#include <iostream>

#include <vector>

#include <list>

#define MAX 55

using namespace std;

int main() {

int adj\_matrix[MAX][MAX], n;

cin >> n;

vector < list < int > > head(n + 1);

for (int i = 1; i <= n; i++)

for (int j = 1; j <= n; j++)

{

cin >> adj\_matrix[i][j];

if (adj\_matrix[i][j])

head[i].push\_front(j);

}

for (int i = 1; i <= n; i++) {

cout << i << " : ";

if (!head[i].empty())

for (auto x : head[i])

cout << x << " ";

cout << endl;

}

return 0;

}

# Tree

# Heap