# Team Notebook

April 25, 2025

## Contents

# 1 A - Introduction

## 1.1 Fatorial

```
//////////////////////
// Fatorial///////////
//////////////////////
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5.040
8! = 40.320
9! = 362.880
10! = 3.628.800
11! = 39.916.800
12! = 479.001.600 [limite do (unsigned) int]
13! = 6.227.020.800
14! = 87.178.291.200
15! = 1.307.674.368.000
16! = 20.922.789.888.000
17! = 355.687.428.096.000
18! = 6.402.373.705.728.000
19! = 121.645.100.408.832.000
20! = 2.432.902.008.176.640.000 [limite do (u)int64_t]
```

## 1.2 Input Time

```
input size      required time complexity
n <= 10               O(n!)
n <= 20               O(2^n)
n <= 500              O(n^3)
n <= 5000             O(n^2)
n <= 10^6             O(n log n) or O(n)
n is large            O(1) or O(log n)
```

## 1.3 Limits

```
////////////////////////////////////
// Limites de representacao de dados
////////////////////////////////////

/* Tipos inteiros:
   Tipo           Bits  Faixa                   Precisao
   -------------- ------ ----------------------- --------
       */
   char           8      0..127                  2
   signed char    8     -128..127                2
   unsigned char  8      0..255                  2
   short          16    -32768..32767            4
   unsigned short 16    0..65535                 4
   int            32    -2*10^9..2*10^9          9
   unsigned int   32    0..4*10^9                9
   int64_t        64    -9*10^18..9*10^18        18
   uint64_t       64    0..18*10^18              19

/* Tipos flutuante:
   Tipo           Bits  Expoente   Precisao
   -------------- ------ ---------- -------- */
   float          32    38         6
   double         64    308        15
   long double    80    19728      18
```

## 1.4 Primes

```
//////////////////////////////////////
// Primos at 10.000 //////////////////
//////////////////////////////////////
[Existem 1.229 nmeros primos at 10.000.]
 2 3 5 7 11 13 17 19 23 29 31
 37 41 43 47 53 59 61 67 71 73 79
 83 89 97 101 103 107 109 113 127 131 137
 139 149 151 157 163 167 173 179 181 191 193
 197 199 211 223 227 229 233 239 241 251 257
 263 269 271 277 281 283 293 307 311 313 317
 331 337 347 349 353 359 367 373 379 383 389
 397 401 409 419 421 431 433 439 443 449 457
 461 463 467 479 487 491 499 503 509 521 523
 541 547 557 563 569 571 577 587 593 599 601
 607 613 617 619 631 641 643 647 653 659 661
 673 677 683 691 701 709 719 727 733 739 743
 751 757 761 769 773 787 797 809 811 821 823
 827 829 839 853 857 859 863 877 881 883 887
 907 911 919 929 937 941 947 953 967 971 977
 983 991 997 1009 1013 1019 1021 1031 1033 1039 1049
 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117
 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213
 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289
 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373
 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453
 1459 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531
 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607
 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693
 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777
 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871
 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951
 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029
 2039 2053 2063 2069 2081 2083 2087 2089 2099 2111 2113
 2129 2131 2137 2141 2143 2153 2161 2179 2203 2207 2213
 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293
 2297 2309 2311 2333 2339 2341 2347 2351 2357 2371 2377
 2381 2383 2389 2393 2399 2411 2417 2423 2437 2441 2447
 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551
 2557 2579 2591 2593 2609 2617 2621 2633 2647 2657 2659
 2663 2671 2677 2683 2687 2689 2693 2699 2707 2711 2713
 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797
 2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887
 2897 2903 2909 2917 2927 2939 2953 2957 2963 2969 2971
 2999 3001 3011 3019 3023 3037 3041 3049 3061 3067 3079
 3083 3089 3109 3119 3121 3137 3163 3167 3169 3181 3187
 3191 3203 3209 3217 3221 3229 3251 3253 3257 3259 3271
 3299 3301 3307 3313 3319 3323 3329 3331 3343 3347 3359
 3361 3371 3373 3389 3391 3407 3413 3433 3449 3457 3461
 3463 3467 3469 3491 3499 3511 3517 3527 3529 3533 3539
 3541 3547 3557 3559 3571 3581 3583 3593 3607 3613 3617
 3623 3631 3637 3643 3659 3671 3673 3677 3691 3697 3701
 3709 3719 3727 3733 3739 3761 3767 3769 3779 3793 3797
 3803 3821 3823 3833 3847 3851 3853 3863 3877 3881 3889
 3907 3911 3917 3919 3923 3929 3931 3943 3947 3967 3989
 4001 4003 4007 4013 4019 4021 4027 4049 4051 4057 4073
 4079 4091 4093 4099 4111 4127 4129 4133 4139 4153 4157
 4159 4177 4201 4211 4217 4219 4229 4231 4241 4243 4253
 4259 4261 4271 4273 4283 4289 4297 4327 4337 4339 4349
 4357 4363 4373 4391 4397 4409 4421 4423 4441 4447 4451
 4457 4463 4481 4483 4493 4507 4513 4517 4519 4523 4547
 4549 4561 4567 4583 4591 4597 4603 4621 4637 4639 4643
 4649 4651 4657 4663 4673 4679 4691 4703 4721 4723 4729
 4733 4751 4759 4783 4787 4789 4793 4799 4801 4813 4817
 4831 4861 4871 4877 4889 4903 4909 4919 4931 4933 4937
 4943 4951 4957 4967 4969 4973 4987 4993 4999 5003 5009
 5011 5021 5023 5039 5051 5059 5077 5081 5087 5099 5101
 5107 5113 5119 5147 5153 5167 5171 5179 5189 5197 5209
 5227 5231 5233 5237 5261 5273 5279 5281 5297 5303 5309
 5323 5333 5347 5351 5381 5387 5393 5399 5407 5413 5417
 5419 5431 5437 5441 5443 5449 5471 5477 5479 5483 5501
 5503 5507 5519 5521 5527 5531 5557 5563 5569 5573 5581
 5591 5623 5639 5641 5647 5651 5653 5657 5659 5669 5683
 5689 5693 5701 5711 5717 5737 5741 5743 5749 5779 5783
 5791 5801 5807 5813 5821 5827 5839 5843 5849 5851 5857
 5861 5867 5869 5879 5881 5897 5903 5923 5927 5939 5953
 5981 5987 6007 6011 6029 6037 6043 6047 6053 6067 6073
 6079 6089 6091 6101 6113 6121 6131 6133 6143 6151 6163
```

```
6173 6197 6199 6203 6211 6217 6221 6229 6247 6257 6263
6269 6271 6277 6287 6299 6301 6311 6317 6323 6329 6337
6343 6353 6359 6361 6367 6373 6379 6389 6397 6421 6427
6449 6451 6469 6473 6481 6491 6521 6529 6547 6551 6553
6563 6569 6571 6577 6581 6599 6607 6619 6637 6653 6659
6661 6673 6679 6689 6691 6701 6703 6709 6719 6733 6737
6761 6763 6779 6781 6791 6793 6803 6823 6827 6829 6833
6841 6857 6863 6869 6871 6883 6899 6907 6911 6917 6947
6949 6959 6961 6967 6971 6977 6983 6991 6997 7001 7013
7019 7027 7039 7043 7057 7069 7079 7103 7109 7121 7127
7129 7151 7159 7177 7187 7193 7207 7211 7213 7219 7229
7237 7243 7247 7253 7283 7297 7307 7309 7321 7331 7333
7349 7351 7369 7393 7411 7417 7433 7451 7457 7459 7477
7481 7487 7489 7499 7507 7517 7523 7529 7537 7541 7547
7549 7559 7561 7573 7577 7583 7589 7591 7603 7607 7621
7639 7643 7649 7669 7673 7681 7687 7691 7699 7703 7717
7723 7727 7741 7753 7757 7759 7789 7793 7817 7823 7829
7841 7853 7867 7873 7877 7879 7883 7901 7907 7919 7927
7933 7937 7949 7951 7963 7993 8009 8011 8017 8039 8053
8059 8069 8081 8087 8089 8093 8101 8111 8117 8123 8147
8161 8167 8171 8179 8191 8209 8219 8221 8231 8233 8237
8243 8263 8269 8273 8287 8291 8293 8297 8311 8317 8329
8353 8363 8369 8377 8387 8389 8419 8423 8429 8431 8443
8447 8461 8467 8501 8513 8521 8527 8537 8539 8543 8563
8573 8581 8597 8599 8609 8623 8627 8629 8641 8647 8663
8669 8677 8681 8689 8693 8699 8707 8713 8719 8731 8737
8741 8747 8753 8761 8779 8783 8803 8807 8819 8821 8831
8837 8839 8849 8861 8863 8867 8887 8893 8923 8929 8933
8941 8951 8963 8969 8971 8999 9001 9007 9011 9013 9029
9041 9043 9049 9059 9067 9091 9103 9109 9127 9133 9137
9151 9157 9161 9173 9181 9187 9199 9203 9209 9221 9227
9239 9241 9257 9277 9281 9283 9293 9311 9319 9323 9337
9341 9343 9349 9371 9377 9391 9397 9403 9413 9419 9421
9431 9433 9437 9439 9461 9463 9467 9473 9479 9491 9497
9511 9521 9533 9539 9547 9551 9587 9601 9613 9619 9623
9629 9631 9643 9649 9661 9677 9679 9689 9697 9719 9721
9733 9739 9743 9749 9767 9769 9781 9787 9791 9803 9811
9817 9829 9833 9839 9851 9857 9859 9871 9883 9887 9901
9907 9923 9929 9931 9941 9949 9967 9973
```

### 1.5 Template

```cpp
#include <bits/stdc++.h>

using namespace std;

#define _ ios::sync_with_stdio(0);cin.tie(0);
#define endl '\n' // CAREFUL INTERACTIVE

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

int main() { _
    cout << fixed << setprecision(0); // remove when working
         with float

    return 0;
}
```

## 2 Dynamic Programming

### 2.1 Fractional Knapsack

```cpp
/*
Given two arrays, val[] and wt[], representing the values
     and weights of items, and an integer capacity
     representing the maximum weight a knapsack can hold,
     the task is to determine the maximum total value that
     can be achieved by putting items in the knapsack. You
     are allowed to break items into fractions if necessary.

Note: Return the maximum value as a double, rounded to 6
     decimal places.

Examples:

Input: val[] = [60, 100, 120], wt[] = [10, 20, 30], capacity
      = 50
Output: 240
Explanation: By taking items of weight 10 and 20 kg and 2/3
     fraction of 30 kg.
Hence total price will be 60+100+(2/3)(120) = 240


Input: val[] = [500], wt[] = [30], capacity = 10
Output: 166.667
*/

// C++ code to implement Fractional Knapsack Problem
#include <bits/stdc++.h>
using namespace std;

// Comparison function to sort items based on value/weight
     ratio
bool compare(vector<int>& a, vector<int>& b) {
    double a1 = (1.0 * a[0]) / a[1];
    double b1 = (1.0 * b[0]) / b[1];
    return a1 > b1;
}

double fractionalKnapsack(vector<int>& val, vector<int>& wt,
      int capacity) {
    int n = val.size();

    // Create 2D vector to store value and weight
    // items[i][0] = value, items[i][1] = weight
    vector<vector<int>> items(n, vector<int>(2));

    for (int i = 0; i < n; i++) {
        items[i][0] = val[i];
        items[i][1] = wt[i];
    }

    // Sort items based on value-to-weight ratio in
         descending order
    sort(items.begin(), items.end(), compare);

    double res = 0.0;
    int currentCapacity = capacity;

    // Process items in sorted order
    for (int i = 0; i < n; i++) {

        // If we can take the entire item
        if (items[i][1] <= currentCapacity) {
            res += items[i][0];
            currentCapacity -= items[i][1];
        }

        // Otherwise take a fraction of the item
        else {
            res += (1.0 * items[i][0] / items[i][1]) *
                 currentCapacity;

            // Knapsack is full
            break;
        }
    }

    return res;
}

int main() {
    vector<int> val = {60, 100, 120};
    vector<int> wt = {10, 20, 30};
```

```
    int capacity = 50;

    cout << fractionalKnapsack(val, wt, capacity) << endl;

    return 0;
}
```

## 2.2 Knapsack 0-1

```
/*
Given n items where each item has some weight and profit
     associated with it and also given a bag with capacity W
     , [i.e., the bag can hold at most W weight in it]. The
     task is to put the items into the bag such that the sum
      of profits associated with them is the maximum
     possible.

Note: The constraint here is we can either put an item
     completely into the bag or cannot put it at all [It is
     not possible to put a part of an item into the bag].

Input: W = 4, profit[] = [1, 2, 3], weight[] = [4, 5, 1]
Output: 3
Explanation: There are two items which have weight less than
      or equal to 4. If we select the item with weight 4,
     the possible profit is 1. And if we select the item
     with weight 1, the possible profit is 3. So the maximum
      possible profit is 3. Note that we cannot put both the
      items with weight 4 and 1 together as the capacity of
     the bag is 4.


Input: W = 3, profit[] = [1, 2, 3], weight[] = [4, 5, 6]
Output: 0
*/
#include <bits/stdc++.h>
using namespace std;

// Function to find the maximum profit
int knapsack(int W, vector<int> &val, vector<int> &wt) {

    // Initializing dp vector
    vector<int> dp(W + 1, 0);

    // Taking first i elements
    for (int i = 1; i <= wt.size(); i++) {

        // Starting from back, so that we also have data of
        // previous computation of i-1 items
        for (int j = W; j >= wt[i - 1]; j--) {
            dp[j] = max(dp[j], dp[j - wt[i - 1]] + val[i -
                1]);
        }
    }
    return dp[W];
}

int main() {
    vector<int> val = {1, 2, 3};
    vector<int> wt = {4, 5, 1};
    int W = 4;

    cout << knapsack(W, val, wt) << endl;
    return 0;
}
```

# 3 Graph

## 3.1 BFS Grid

```
#include <bits/stdc++.h>

using namespace std;

// BFS COM PRED - LABYRINTH CSES

vector<vector<char>> G;
vector<vector<int>> visited;
vector<pair<int, int>> moves = {{-1, 0}, {1, 0}, {0, -1},
    {0, 1}};
vector<vector<pair<int, int>>> previous;

void bfs(int i, int j) {
    queue<pair<int, int>> q;
    q.push({i, j});

    while (!q.empty()) {
        pair<int, int> p = q.front();
        q.pop();

        if (visited[p.first][p.second])
            continue;

        if (G[p.first][p.second] == 'B') {
            return;
        }

        visited[p.first][p.second] = 1;

        for (pair<int, int> pp : moves) {
            int I = p.first + pp.first, J = p.second + pp.
                second;

            if (I < 0 || J < 0 || I >= G.size() || J >= G[0].
                size() || visited[I][J] || G[I][J] == '#')
                continue;

            previous[I][J] = {p.first, p.second};
            q.push({I, J});
        }
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin >> n >> m;

    G = vector<vector<char>>(n, vector<char>(m));
    visited = vector<vector<int>>(n, vector<int>(m, 0));
    previous = vector<vector<pair<int, int>>>(n, vector<pair<
        int, int>>(m));

    int beginI = 0, beginJ = 0, finalI = 0, finalJ = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> G[i][j];
            if (G[i][j] == 'A') {
                beginI = i;
                beginJ = j;
            }

            if (G[i][j] == 'B') {
                finalI = i;
                finalJ = j;
            }
        }
    }

    previous[finalI][finalJ] = {INT_MIN, INT_MAX};

    bfs(beginI, beginJ);
```

```
    if (previous[finalI][finalJ].first == INT_MIN && previous
         [finalI][finalJ].second == INT_MAX) {
        cout << "NO" << "\n";
        return 0;
    }

    cout << "YES\n";

    stack<char> ans;

    while (finalI != beginI || finalJ != beginJ) {
        pair<int, int> &pre = previous[finalI][finalJ];

        if (finalI < pre.first)
            ans.push('U');
        else if (finalI > pre.first)
            ans.push('D');
        else if (finalJ < pre.second)
            ans.push('L');
        else
            ans.push('R');

        finalI = pre.first;
        finalJ = pre.second;
    }

    cout << ans.size() << '\n';

    while (!ans.empty()) {
        cout << ans.top();
        ans.pop();
    }

    return 0;
}
```

## 3.2 BFS

```
#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> G;
vector<int> visited;
vector<int> pred;

// BFS COM PREDECESSOR LISTA ADJ

void bfs(int i) {
    queue<int> q;
    q.push(i);

    while (!q.empty()) {
        int a = q.front();
        q.pop();

        if (visited[a])
            continue;

        visited[a] = 1;

        for (int p : G[a]) {
            if (!visited[p]) {
                q.push(p);
                if (pred[p] == -1)
                    pred[p] = a;
            }
        }
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout << fixed << setprecision(0);

    int n, m;
    cin >> n >> m;

    G = vector<vector<int>>(n + 1, vector<int>());
    visited = vector<int>(n + 1, 0);
    pred = vector<int> (n + 1, -1);

    int u, v;
    for (int i = 0; i < m; i++) {
        cin >> u >> v;
        G[u].push_back(v);
        G[v].push_back(u);
    }

    bfs(1);

    if (!visited[n]) {
        cout << "IMPOSSIBLE\n";
        return 0;
    }

    int p = pred[n];
    stack<int> s;
    s.push(n);

    while (p != -1) {
        s.push(p);
        p = pred[p];
    }

    cout << s.size() << '\n';

    while (!s.empty()) {
        cout << s.top() << ' ';
        s.pop();
    }

    cout << '\n';

    return 0;
}
```

## 3.3 Bellman-Ford

```
// Given a weighted graph with V vertices and E edges, along
     with a source vertex src, the task is to compute the
     shortest distances from the source to all other
     vertices. If a vertex is unreachable from the source,
     its distance should be marked as 108. In the presence
     of a negative weight cycle, return -1 to signify that
     shortest path calculations are not feasible.

#include <iostream>
#include <vector>
using namespace std;

vector<int> bellmanFord(int V, vector<vector<int>>& edges,
     int src) {

    // Initially distance from source to all
    // other vertices is not known(Infinite).
 vector<int> dist(V, 1e8);
 dist[src] = 0;

 // Relaxation of all the edges V times, not (V - 1) as we
    // need one additional relaxation to detect negative
        cycle
 for (int i = 0; i < V; i++) {

  for (vector<int> edge : edges) {
   int u = edge[0];
   int v = edge[1];
```

```
   int wt = edge[2];
   if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {

               // If this is the Vth relaxation, then there
                    is
               // a negative cycle
               if(i == V - 1)
                   return {-1};

               // Update shortest distance to node v
               dist[v] = dist[u] + wt;
           }
  }
 }

   return dist;
}

int main() {

    // Number of vertices in the graph
    int V = 5;

    // Edge list representation: {source, destination, weight
        }
    vector<vector<int>> edges = {
        {1, 3, 2},
        {4, 3, -1},
        {2, 4, 1},
        {1, 2, 1},
        {0, 1, 5}
    };

    // Define the source vertex
    int src = 0;

    // Run Bellman-Ford algorithm to get shortest paths from
        src
    vector<int> ans = bellmanFord(V, edges, src);

    // Output the shortest distances from src to all vertices
    for (int dist : ans)
        cout << dist << " ";

    return 0;
}
```

## 3.4 DFS Grid

```
#include <bits/stdc++.h>

using namespace std;

vector<vector<char>> G;
vector<vector<int>> visited;
vector<pair<int, int>> moves = {{-1, 0}, {0, -1}, {0, 1},
    {1, 0}};

void dfs(int i, int j) {
    if (visited[i][j])
        return;

    visited[i][j] = 1;

    for (pair<int, int> p : moves) {
        int I = i + p.first, J = j + p.second;

        if (I < 0 || J < 0 || I >= G.size() || J >= G[0].size
            ())
            continue;

        dfs(I, J);
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin >> n >> m;

    G = vector<vector<char>>(n, vector<char>(m));
    visited = vector<vector<int>>(n, vector<int>(m, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> G[i][j];
        }
    }

    return 0;
}
```

## 3.5 DFS

```
#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> G;
vector<int> visited;

void dfs(int city) {
    stack<int> st;
    st.push(city);

    while (!st.empty()) {
        int c = st.top();
        st.pop();

        if (visited[c])
            continue;

        visited[c] = 1;

        for (int p : G[c]) {
            if (!visited[p])
                st.push(p);
        }
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin >> n >> m;

    G = vector<vector<int>>(n + 1, vector<int>());

    int a, b;
    for (int i = 0; i < m; i++) {
        cin >> a >> b;
        G[a].push_back(b);
        G[b].push_back(a);
    }

    int total = 0;

    string ans;
    visited = vector<int>(n + 1, 0);

    return 0;
}
```

## 3.6 Djikstra

```cpp
#include <bits/stdc++.h>

using namespace std;

vector<vector<pair<int, int>>> G;
vector<int> visited;
vector<int> dist;

void dijkstra(int i) {
    priority_queue<pair<int, int>> q;
    dist[i] = 0;
    q.push({0, i});

    while (!q.empty()) {
        int a = q.top().second;
        q.pop();

        if (visited[a])
            continue;

        visited[a] = 1;

        for (pair<int, int> p : G[a]) {
            int b = p.first, w = p.second;
            if (dist[a] + w < dist[b]) {
                dist[b] = dist[a] + w;
                q.push({ -dist[b], b });
            }
        }
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout << fixed << setprecision(0); // remove when working
         with floating point

    int n, m;
    cin >> n >> m;

    G = vector<vector<pair<int, int>>> (n + 1, vector<pair<
         int ,int>>());
    visited = vector<int> (n + 1, 0);
    dist = vector<int> (n + 1, INFINITY);

    int s, t, b;
    while (m--) {
        cin >> s >> t >> b;
        G[s].push_back({t, b});
        //G[t].push_back({s, b});
    }

    dijkstra(0);

    cout << dist[n + 1] << '\n';

    return 0;
}
```

## 3.7 Floyd-Warshall

```cpp
//Given a matrix dist[][] of size n x n, where dist[i][j]
     represents the weight of the edge from node i to node j
     . If there is no direct edge, dist[i][j] is set to a
     large value (e.g., 10) to represent infinity. The
     diagonal entries dist[i][i] are 0, since the distance
     from a node to itself is zero. The graph may contain
     negative edge weights, but it does not contain any
     negative weight cycles.
//Your task is to determine the shortest path distance
     between all pair of nodes i and j in the graph.

#include <bits/stdc++.h>

using namespace std;

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall(vector<vector<int>> &dist) {
    int V = dist.size();

    // Add all vertices one by one to
    // the set of intermediate vertices.
    for (int k = 0; k < V; k++) {

        // Pick all vertices as source one by one
        for (int i = 0; i < V; i++) {

            // Pick all vertices as destination
            // for the above picked source
            for (int j = 0; j < V; j++) {

                // shortest path from
                // i to j
                if(dist[i][k] != 1e8 && dist[k][j]!= 1e8)
                    dist[i][j] = min(dist[i][j],dist[i][k] + dist[
                        k][j]);
            }
        }
    }
}

int main() {
    int INF = 100000000;
    vector<vector<int>> dist = {
        {0, 4, INF, 5, INF},
        {INF, 0, 1, INF, 6},
        {2, INF, 0, 3, INF},
        {INF, INF, 1, 0, 2},
        {1, INF, INF, 4, 0}
    };

    floydWarshall(dist);
    for(int i = 0; i<dist.size(); i++) {
        for(int j = 0; j<dist.size(); j++) {
            cout<<dist[i][j]<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

## 3.8 Maximum Flow

```cpp
//The max flow problem is a classic optimization problem in
     graph theory that involves finding the maximum amount
     of flow that can be sent through a network of pipes,
     channels, or other pathways, subject to capacity
     constraints. The problem can be used to model a wide
     variety of real-world situations, such as
     transportation systems, communication networks, and
     resource allocation.

#include <bits/stdc++.h>

using namespace std;

#define _ ios::sync_with_stdio(0);cin.tie(0);

vector<vector<int>> capacity;
vector<vector<int>> G;

int bfs(int s, int t, vector<int> &parent) {
 fill(parent.begin(), parent.end(), -1);
```

```
 parent[s] = -2;
 queue<pair<int, int>> q;
 q.push({s, INT_MAX});

 while (!q.empty()) {
  int cur = q.front().first;
  int flow = q.front().second;
  q.pop();

  for (int next : G[cur]) {
   if (parent[next] == -1 && capacity[cur][next]) {
    parent[next] = cur;
    int new_flow = min(flow, capacity[cur][next]);
    if (next == t)
     return new_flow;
    q.push({ next, new_flow });
   }
  }
 }

 return 0;
}

int maxflow(int s, int t) {
 int flow = 0;
 vector<int> parent(G.size());
 int new_flow;

 while (new_flow = bfs(s, t, parent)) {
  flow += new_flow;
  int cur = t;
  while (cur != s) {
   int prev = parent[cur];
   capacity[prev][cur] -= new_flow;
   capacity[cur][prev] += new_flow;
   cur = prev;
  }
 }

 return flow;
}

int main() { _
 int n, m;
 cin >> n >> m;
 int s, t;
 cin >> s >> t;

 G = vector<vector<int>>(n + 1, vector<int>());
 capacity = vector<vector<int>>(n+1, vector<int>(n+1, 0));

 int u, v, w;
 while (m--) {
  cin >> u >> v >> w;
  capacity[u][v] += w;
  G[u].push_back(v);
  G[v].push_back(u);
 }

 cout << maxflow(s, t) << "\n";

 return 0;
}
```

## 3.9 Minimum Spanning Tree

```
#include <bits/stdc++.h>

using namespace std;

// Disjoint set data struture
class DSU {
    vector<int> parent, rank;

public:
    DSU(int n) {
        parent.resize(n);
        rank.resize(n);
        for (int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 1;
        }
    }

    int find(int i) {
        return (parent[i] == i) ? i : (parent[i] = find(
             parent[i]));
    }

    void unite(int x, int y) {
        int s1 = find(x), s2 = find(y);
        if (s1 != s2) {
            if (rank[s1] < rank[s2]) parent[s1] = s2;
            else if (rank[s1] > rank[s2]) parent[s2] = s1;
            else parent[s2] = s1, rank[s1]++;
        }
    }
};
bool comparator(vector<int> &a,vector<int> &b){
    if(a[2]<=b[2])return true;
    return false;
}
// V = vertices on graph
int kruskalsMST(int V, vector<vector<int>> &edges) {

    // Sort all edhes
    sort(edges.begin(), edges.end(),comparator);

    // Traverse edges in sorted order
    DSU dsu(V);
    int cost = 0, count = 0;

    for (auto &e : edges) {
        int x = e[0], y = e[1], w = e[2];

        // Make sure that there is no cycle
        if (dsu.find(x) != dsu.find(y)) {
            dsu.unite(x, y);
            cost += w;
            if (++count == V - 1) break;
        }
    }
    return cost;
}

int main() {

    // An edge contains, weight, source and destination
    vector<vector<int>> edges = {
        {0, 1, 10}, {1, 3, 15}, {2, 3, 4}, {2, 0, 6}, {0, 3,
             5}
    };

    cout<<kruskalsMST(4, edges);

    return 0;
}
```

## 3.10 Segment Tree

```
/*
Let us consider the following problem to understand Segment
     Trees without recursion.
We have an array arr[0 . . . n-1]. We should be able to,

Find the sum of elements from index l to r where 0 <= l <= r
      <= n-1
```

```
Change the value of a specified element of the array to a
     new value x. We need to do arr[i] = x where 0 <= i <= n
     -1.
*/

#include <bits/stdc++.h>
using namespace std;

// limit for array size
const int N = 100000;

int n; // array size

// Max size of tree
int tree[2 * N];

// function to build the tree
void build( int arr[])
{
    // insert leaf nodes in tree
    for (int i=0; i<n; i++)
        tree[n+i] = arr[i];

    // build the tree by calculating parents
    for (int i = n - 1; i > 0; --i)
        tree[i] = tree[i<<1] + tree[i<<1 | 1];
}

// function to update a tree node
void updateTreeNode(int p, int value)
{
    // set value at position p
    tree[p+n] = value;
    p = p+n;

    // move upward and update parents
    for (int i=p; i > 1; i >>= 1)
        tree[i>>1] = tree[i] + tree[i^1];
}

// function to get sum on interval [l, r)
int query(int l, int r)
{
    int res = 0;

    // loop to find the sum in the range
    for (l += n, r += n; l < r; l >>= 1, r >>= 1)
    {
        if (l&1)
            res += tree[l++];

        if (r&1)
            res += tree[--r];
    }

    return res;
}

// driver program to test the above function
int main()
{
    int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

    // n is global
    n = sizeof(a)/sizeof(a[0]);

    // build tree
    build(a);

    // print the sum in range(1,2) index-based
    cout << query(1, 3)<<endl;

    // modify element at 2nd index
    updateTreeNode(2, 1);

    // print the sum in range(1,2) index-based
    cout << query(1, 3)<<endl;

    return 0;
}
```

## 3.11 Topological Sort

```
#include <bits/stdc++.h>

using namespace std;

// TOPOLOGICAL SORT FROM COURSE SCHEDULE (CSES) USING DFS
// IT ALREADY CHECKS IF THERE IS A CYCLE

vector<vector<int>> G;
vector<int> visited;
vector<int> visitedPath;
stack<int> topological;

void dfs(int i, int path, int &ok) {
    if (visited[i])
        return;
    visited[i] = 1;
    visitedPath[i] = path;

    for (int p : G[i]) {
        if (visited[p] && visitedPath[p]) {
            ok = 0;
            return;
        }

        if (!visited[p]) {
            dfs(p, path, ok);
        }
    }

    visitedPath[i] = 0;
    topological.push(i);
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin >> n >> m;

    G = vector<vector<int>>(n + 1, vector<int>());
    visited = vector<int>(n + 1, 0);
    visitedPath = vector<int> (n + 1, 0);

    int a, b;
    for (int i = 0; i < m; i++) {
        cin >> a >> b;
        G[a].push_back(b);
    }

    for (int i = 1; i <= n; i++) {
        if (visited[i])
            continue;

        int ok = 1;
        dfs(i, i, ok);

        if (!ok) {
            cout << "IMPOSSIBLE\n";
            return 0;
        }
    }

    while (!topological.empty()) {
        int c = topological.top();
```

```
        topological.pop();

        cout << c << ' ';
    }

    return 0;
}
```

## 3.12 Union-Find

```
/*
1. Find:
The task is to find representative of the set of a given
     element. The representative is always root of the tree.
      So we implement find() by recursively traversing the
     parent array until we hit a node that is root (parent
     of itself).

2. Union:
The task is to combine two sets and make one. It takes two
     elements as input and finds the representatives of
     their sets using the Find operation, and finally puts
     either one of the trees (representing the set) under
     the root node of the other tree.
*/

#include <iostream>
#include <vector>
using namespace std;

class UnionFind {
    vector<int> parent;
public:
    UnionFind(int size) {

        parent.resize(size);

        // Initialize the parent array with each
        // element as its own representative
        for (int i = 0; i < size; i++) {
            parent[i] = i;
        }
    }

    // Find the representative (root) of the
    // set that includes element i
    int find(int i) {

        // If i itself is root or representative
        if (parent[i] == i) {
            return i;
        }

        // Else recursively find the representative
        // of the parent
        return find(parent[i]);
    }

    // Unite (merge) the set that includes element
    // i and the set that includes element j
    void unite(int i, int j) {

        // Representative of set containing i
        int irep = find(i);

        // Representative of set containing j
        int jrep = find(j);

        // Make the representative of i's set
        // be the representative of j's set
        parent[irep] = jrep;
    }
};

int main() {
    int size = 5;
    UnionFind uf(size);
    uf.unite(1, 2);
    uf.unite(3, 4);
    bool inSameSet = (uf.find(1) == uf.find(2));
    cout << "Are 1 and 2 in the same set? "
         << (inSameSet ? "Yes" : "No") << endl;
    return 0;
}
```

# 4 Math

## 4.1 Divisors

```
#include <bits/stdc++.h>

using namespace std;

int main() {
   ios_base::sync_with_stdio(false);
   cin.tie(0);

   long long n;
   cin >> n;

   for (int i = 1; i <= sqrt(n); i++) {
       if (n % i == 0) {
           if (n / i == i) {
               cout << i << '\n';
           } else {
               cout << i << " " << n / i << '\n';
           }
       }
   }

   return 0;
}
```

## 4.2 GCD

```
#include <bits/stdc++.h>

using namespace std;

int GCD(int a, int b) {
    if (a == 0)
        return b;

    return GCD(b%a, a);
}
```

## 4.3 Lines Intersection

```
// Encontra  interseo  de duas retas descritas por ax+by=c.
#include <bits/stdc++.h>
 using namespace std;

 int main() {
     double a1, b1, c1, a2, b2, c2;
    cin >> a1 >> b1 >> c1 >> a2 >> b2 >> c2;
    double det = a1*b2 - a2*b1;
    if (fabs(det) < 1e-9) {
        cout << "Paralelas";
    } else {
        double x = (b2*c1 - b1*c2) / det;
        double y = (a1*c2 - a2*c1) / det;
        cout << x << " " << y;
    }
    return 0;
```

```
}
```

## 4.4 Log A to base B

```
#include <bits/stdc++.h>

using namespace std;

long long log_a_to_base_b(long long a, long long b) {
    return log2(a) / log2(b);
}
```

## 4.5 Primality Test

```
// Optimised school method based C++ program to check if a
// number is prime
#include <bits/stdc++.h>
using namespace std;

bool isPrime(int n)
{
    // Corner case
    if (n <= 1)
        return false;

    // Check from 2 to square root of n
    for (int i = 2; i <= sqrt(n); i++)
        if (n % i == 0)
            return false;

    return true;
}

// Driver Program to test above function
int main()
{
    isPrime(11) ? cout << " true\n" : cout << " false\n";
    isPrime(15) ? cout << " true\n" : cout << " false\n";
    return 0;
}

// This code is contributed by Vikash Sangai
```

## 4.6 Prime Factorization

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
 int n;
 cin >> n;

 int spf[n + 1];

 for (int i = 2; i <= n; i++)
  spf[i] = i;

 for (int x = 2; x <= n; x++) // Calcula o vetor spf
 {
  if (spf[x] != x)
   continue;

  for (int i = 2*x; i <= n; i += x)
   spf[i] = min(spf[i], x);
 }

 int value = n;
 vector<int> primes; // Armazena a fatorao prima de n

 while (value != 1) // Itera enquanto ele tiver um fator
      primo
 {
  primes.push_back(spf[value]);
  value /= spf[value]; // Divide value pelo seu menor fator
       primo
 }

 cout << " Fatorao  prima de " << n << ": " << primes[0];

 for (int i = 1; i < (int)primes.size(); i++)
  cout << "*" << primes[i];
}
```

## 4.7 Sieve

```
#include <bits/stdc++.h>

using namespace std;

vector<bool> primes;

void sieve(long long n) {
    primes = vector<bool>(n + 1, true);
    primes[0] = primes[1] = false;

    for (int i = 2; i <= sqrt(n); i++) {
        if (primes[i]) {
            for (int j = i * i; j <= n; j += i) {
                primes[j] = false;
            }
        }
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    sieve(100);

    return 0;
}
```

# 5 String

## 5.1 Knuth-Morris-Prat

```
/*
Given two strings txt and pat, the task is to return all
     indices of occurrences of pat within txt.

Examples:

Input: txt =   abcab  ,  pat =   ab
Output: [0, 3]
Explanation: The string   ab   occurs twice in txt, first
     occurrence starts from index 0 and second from index 3.


Input: txt=   aabaacaadaabaaba  , pat =   aaba
Output: [0, 9, 12]
*/

// C++ program to search the pattern in given text using
// KMP Algorithm

#include <iostream>
#include <string>
#include <vector>
```

```cpp
using namespace std;

void constructLps(string &pat, vector<int> &lps) {

    // len stores the length of longest prefix which
    // is also a suffix for the previous index
    int len = 0;

    // lps[0] is always 0
    lps[0] = 0;

    int i = 1;
    while (i < pat.length()) {

        // If characters match, increment the size of lps
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        }

        // If there is a mismatch
        else {
            if (len != 0) {

                // Update len to the previous lps value
                // to avoid reduntant comparisons
                len = lps[len - 1];
            }
            else {

                // If no matching prefix found, set lps[i] to
                    0
                lps[i] = 0;
                i++;
            }
        }
    }
}

vector<int> search(string &pat, string &txt) {
    int n = txt.length();
    int m = pat.length();

    vector<int> lps(m);
    vector<int> res;

    constructLps(pat, lps);

    // Pointers i and j, for traversing
    // the text and pattern
    int i = 0;
    int j = 0;

    while (i < n) {

        // If characters match, move both pointers forward
        if (txt[i] == pat[j]) {
            i++;
            j++;

            // If the entire pattern is matched
            // store the start index in result
            if (j == m) {
                res.push_back(i - j);

                // Use LPS of previous index to
                // skip unnecessary comparisons
                j = lps[j - 1];
            }
        }

        // If there is a mismatch
        else {

            // Use lps value of previous index
            // to avoid redundant comparisons
            if (j != 0)
                j = lps[j - 1];
            else
                i++;
        }
    }
    return res;
}

int main() {
    string txt = "aabaacaadaabaaba";
    string pat = "aaba";

    vector<int> res = search(pat, txt);
    for (int i = 0; i < res.size(); i++)
        cout << res[i] << " ";

    return 0;
}
```

## 5.2 Levenshtein's Distance

```cpp
// Calcula o nmero mnimo de operaes necessrias para
    transformar uma string em outra.
//Levenshtein distance is a measure of the similarity
    between two strings, which takes into account the
    number of insertion, deletion and substitution
    operations needed to transform one string into the
    other.

// C++ code for the above approach:
#include <bits/stdc++.h>
using namespace std;

int levenshteinTwoMatrixRows(const string& str1,
                             const string& str2)
{
    int m = str1.length();
    int n = str2.length();

    vector<int> prevRow(n + 1, 0);
    vector<int> currRow(n + 1, 0);

    for (int j = 0; j <= n; j++) {
        prevRow[j] = j;
    }

    for (int i = 1; i <= m; i++) {
        currRow[0] = i;

        for (int j = 1; j <= n; j++) {
            if (str1[i - 1] == str2[j - 1]) {
                currRow[j] = prevRow[j - 1];
            }
            else {
                currRow[j] = 1
                             + min(

                                 // Insert
                                 currRow[j - 1],
                                 min(

                                     // Remove
                                     prevRow[j],

                                     // Replace
                                     prevRow[j - 1]));
            }
        }

        prevRow = currRow;
    }
```

```
    return currRow[n];
}

// Drivers code
int main()
{
    string str1 = "kitten";
    string str2 = "sitting";

    // Function Call
    int distance = levenshteinTwoMatrixRows(str1, str2);
    cout << "Levenshtein Distance: " << distance;
    return 0;
}
```

## 5.3 Longest Common Subsequence

```
/*
Given two strings, s1 and s2, the task is to find the length
      of the Longest Common Subsequence. If there is no
     common subsequence, return 0. A subsequence is a string
      generated from the original string by deleting 0 or
     more characters, without changing the relative order of
      the remaining characters.

Example:
Input: s1 =   AGGTAB  , s2 =   GXTXAYB
Output: 4
Explanation: The longest common subsequence is GTAB.
*/

// C++ program to find the longest common subsequence of two
      strings
// using space optimization

#include <iostream>
#include <vector>
using namespace std;

int lcs(string &s1, string &s2) {

    int m = s1.length(), n = s2.length();

    // dp vector is initialized to all zeros
    // This vector stores the LCS values for the current row.
    // dp[j] represents LCS of s1[0..i] and s2[0..j]
    vector<int> dp(n + 1, 0);

    // i and j represent the lengths of s1 and s2
        respectively
    for (int i = 1; i <= m; ++i) {

        // prev stores the value from the previous
        // row and previous column (i-1), (j -1)
        // Used to keep track of LCS[i-1][j-1] while updating
             dp[j]
        int prev = dp[0];

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

            // temp temporarily stores the current
            // dp[j] before it gets updated
            int temp = dp[j];

            // If characters match, add 1 to the value
            // from the previous row and previous column
            // dp[j] = 1 + LCS[i-1][j-1]
            if (s1[i - 1] == s2[j - 1])
                dp[j] = 1 + prev;
            else
                // Otherwise, take the maximum of the
                // left (dp[j-1]) and top (dp[j]) values
                dp[j] = max(dp[j - 1], dp[j]);

            // Update prev for the next iteration
            // This keeps the value of the previous
          // row (i-1) for future comparisons
            prev = temp;
        }
    }

    // The last element of the vector contains the length of
        the LCS
    // dp[n] stores the length of LCS of s1[0..m] and s2[0..n
        ]
    return dp[n];
}

int main() {
    string s1 = "AGGTAB", s2 = "GXTXAYB";
    cout << lcs(s1, s2);
    return 0;
}
```

# 6 Vector

## 6.1 Binary Search

```
#include <bits/stdc++.h>

using namespace std;

int binarySeach(vector<int> &arr, int val) {
    int l = 0, r = arr.size() - 1;

    while (l <= r) {
        int middle = (l + r) / 2;

        if (arr[middle] == val)
            return middle;

        if (arr[middle] < val)
            l = middle + 1;
        else
            r = middle - 1;
    }

    return -1; // elemento nao encontrado
}
```

## 6.2 Kadane

```
#include <bits/stdc++.h>

using namespace std;

int main() {
   ios_base::sync_with_stdio(false);
   cin.tie(0);

   vector<int> arr = {3, -2, 1, 5, -4, 7, 9};

   int MAX = arr[0], sum = arr[0];

   for (int i = 1; i < arr.size(); i++) {
       sum = max(arr[i], sum + arr[i]);
       MAX = max(MAX, sum);
   }

   cout << MAX << '\n';

   return 0;
```

```
}
```

## 6.3 Longest Increasing Subsequence

```
/*
Given an array arr[] of size N, the task is to find the
     length of the Longest Increasing Subsequence (LIS) i.e
     ., the longest possible subsequence in which the
     elements of the subsequence are sorted in increasing
     order, in O(N log N).

Examples:

Input: arr[] = {3, 10, 2, 1, 20}
Output: 3
Explanation: The longest increasing subsequence is 3, 10, 20


Input: arr[] = {3, 2}
Output:1
Explanation: The longest increasing subsequences are {3} and
      {2}


Input: arr[] = {50, 3, 10, 7, 40, 80}
Output: 4
Explanation: The longest increasing subsequence is {3, 7,
     40, 80}
*/

// Binary Search Approach of Finding LIS by
// reducing the problem to longest
// common Subsequence
#include <bits/stdc++.h>
using namespace std;

int lengthOfLIS(vector<int>& arr)
{

    // Binary search approach
    int n = arr.size();
    vector<int> ans;

    // Initialize the answer vector with the
    // first element of arr
    ans.push_back(arr[0]);

    for (int i = 1; i < n; i++) {
        if (arr[i] > ans.back()) {

            // If the current number is greater
            // than the last element of the answer
            // vector, it means we have found a
            // longer increasing subsequence.
            // Hence, we append the current number
            // to the answer vector.
            ans.push_back(arr[i]);
        }
        else {

            // If the current number is not
            // greater than the last element of
            // the answer vector, we perform
            // a binary search to find the smallest
            // element in the answer vector that
            // is greater than or equal to the
            // current number.

            // The lower_bound function returns
            // an iterator pointing to the first
            // element that is not less than
            // the current number.
            int low = lower_bound(ans.begin(), ans.end(),
                                  arr[i])
                      - ans.begin();

            // We update the element at the
            // found position with the current number.
            // By doing this, we are maintaining
            // a sorted order in the answer vector.
            ans[low] = arr[i];
        }
    }

    // The length of the answer vector
    // represents the length of the
    // longest increasing subsequence.
    return ans.size();
}

// Driver program to test above function
int main()
{
    vector<int> arr = { 10, 22, 9, 33, 21, 50, 41, 60 };
    // Function call
    printf("Length of LIS is %d\n", lengthOfLIS(arr));
    return 0;
}
```

## 6.4 Prefix Sum

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int n, q;
    cin >> n >> q;

    vector<long long> arr(n + 1);
    arr[0] = 0;
    long long sum = 0, temp;

    for (int i = 1; i <= n; i++) {
        cin >> temp;
        sum += temp;
        arr[i] = sum;
    }

    int a, b;
    while (q--) {
        cin >> a >> b;
        cout << arr[b] - arr[a - 1] << '\n';
    }

    return 0;
}
```

## 6.5 Range Xor Queries

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, q;
    cin >> n >> q;

    vector<long long> arr(n + 1);

    long long sum = 0, temp;
```

```
for (int i = 1; i <= n; i++) {
    cin >> temp;
    sum = sum ^ temp;
    arr[i] = sum;
}
```

```
int a, b;
while (q--) {
    cin >> a >> b;

    cout << (arr[b] ^ arr[a - 1]) << '\n';
}
```

```
    return 0;
}
```