AdaByron 2018 Universidad Rey Juan Carlos -RaspuTeam

Matemáticas y Combinatoria

1. Algoritmo de Euclides (MCM,LCM) (C++) 1
2. Sistemas de ecuaciones lineales, matriz inversa,determinantes 1
3. Números Catalanes (C++) 2
4. Karatsuba (C++) 2
5. Coeficiente Binomial (C++) 3
6. Modulo en Factorial (C++) 3

Grafos

1. Dijkstra's (C++) 3
2. Kruskal's (C++) 4
3. DFS (Ciclos, Componentes Conexas) y BFS (C++) 4
4. Strongly connected components (C++) 5
5. Floyd Warshall(C++) 6
6. Prim (C++) 6
7. BellMan Ford (C++) 7
8. Topological Sort (C++) 7

Estructura de Datos

1. Trie (C++) 7
2. Next Permutations (C++) 8
3. Árboles (C++) 8

Mix

1. Longest Increasing Subsequence (C++) 9
2. Knuth-Morris-Pratt (C++) 9
3. Búsqueda binaria (C++) 9
4. Números primos (C++) 9
5. Otras constantes (C++) 9
6. Partitions Integer (C++) 10
7. Longest common Subsequence (C++) 10
8. Longest increasing common Subsequence (C++) 10

Kruskal: 6 7 1 2 1 1 3 1 3 6 2 3 4 3 6 4 4 6 5 6 5 4 1 -> 8
Topo: 5 5 1 3 1 2 2 3 2 4 3 5 -> 1 2 4 3 5
Bipartito: color a -1 posible da la solución

Algoritmo de Euclides (MCM,LCM) (C++)

```
int gcd(int a, int b) {
while (b > 0) {
int temp = b; b = a % b;  a = temp;  }
    return a; }
int lcm(int a, int b){ return a*(b/gcd(a,b));}
```

Sistemas de ecuaciones lineales, inversa, etc (C++)

```
// (1) solving systems of linear equations (AX=B)
// (2) inverting matrices (AX=I)
// (3) computing determinants of square matrices
// INPUT: a[][] = an nxn matrix
// b[][] = an nxm matrix
// OUTPUT: X = an nxm matrix (stored in b[][])
// A^{-1} = an nxn matrix (stored in a[][])
// returns determinant of a[][]
const double EPS = 1e-10;
typedef vector<int> VI; typedef double T;
typedef vector<T> VT; typedef vector<VT> VVT;
T GaussJordan(VVT &a, VVT &b) {
const int n = a.size(); const int m = b[0].size();
VI irow(n), icol(n), ipiv(n); T det = 1;
for (int i = 0; i < n; i++) {
int pj = -1, pk = -1;
for (int j = 0; j < n; j++) if (!ipiv[j])
for (int k = 0; k < n; k++) if (!ipiv[k])
if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) {
 pj = j; pk = k; }
if (fabs(a[pj][pk]) < EPS) { exit(0); }
ipiv[pk]++; swap(a[pj], a[pk]); swap(b[pj], b[pk]);
if (pj != pk) det *= -1; irow[i] = pj; icol[i] = pk;
T c = 1.0 / a[pk][pk];
det *= a[pk][pk]; a[pk][pk] = 1.0;
for (int p = 0; p < n; p++) a[pk][p] *= c;
for (int p = 0; p < m; p++) b[pk][p] *= c;
for (int p = 0; p < n; p++) if (p != pk) {
c = a[p][pk]; a[p][pk] = 0;
for (int q=0; q<n; q++) a[p][q]-=a[pk][q]*c;
for (int q=0; q<m; q++) b[p][q]-=b[pk][q]*c; } }
```

```
for (int p = n-1; p>=0; p--) if (irow[p]!=icol[p]){
for (int k = 0; k < n; k++) swap(a[k][irow[p]],
a[k][icol[p]]); } return det; }
int main() {
const int n = 4; const int m = 2;
double A[n][n] = {
{1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
VVT a(n), b(n);
for (int i = 0; i < n; i++) {
a[i] = VT(A[i], A[i] + n);
b[i] = VT(B[i], B[i] + m); }
double det = GaussJordan(a, b); // expected: 60
cout << "Determinant: " << det << endl;
// expected: -0.233333 0.166667 0.133333 0.0666667
// 0.166667 0.166667 0.333333 -0.333333
// 0.233333 0.833333 -0.133333 -0.0666667
// 0.05 -0.75 -0.1 0.2
cout << "Inverse: " << endl;
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++)
cout << a[i][j] << ' '; cout << endl; }
// expected: 1.63333 1.3
// -0.166667 0.5 // 2.36667 1.7 // -1.85 -1.35
for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++)
cout << b[i][j] << ' '; cout << endl; }}
```

Números Catalanes (C++)

```
// 1 1 2 5 14 42 132 429 1430 4862 16796 58786
unsigned long int catalanDP(unsigned int n){
unsigned long int catalan[n+1];
catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++){ catalan[i] = 0;
        for (int j=0; j<i; j++)
            catalan[i]+=catalan[j]*catalan[i-j-1];
    } return catalan[n]; }
int main() { for (int i = 0; i < 10; i++)
cout << catalanDP(i) << " "; return 0; }
```

Karatsuba (C++)

```
int makeEqualLength(string &str1, string &str2){
    int len1 = str1.size(); int len2 = str2.size();
    if (len1 < len2){
        for (int i = 0 ; i < len2 - len1 ; i++)
            str1 = '0' + str1; return len2;}
    else if (len1 > len2) {
        for (int i = 0 ; i < len1 - len2 ; i++)
            str2 = '0' + str2; } return len1;}
string addBitStrings( string first, string second ){
    string result;
    int length = makeEqualLength(first, second);
    int carry = 0;  // Initialize carry
    for (int i = length-1 ; i >= 0 ; i--) {
        int firstBit = first.at(i) - '0';
        int secondBit = second.at(i) - '0';
        int sum = (firstBit ^ secondBit ^ carry)+'0';
        result = (char)sum + result;
        carry = (firstBit&secondBit) |
(secondBit&carry) | (firstBit&carry);  }
    if (carry)  result = '1' + result;
    return result; }
int multiplyiSingleBit(string a, string b)
{  return (a[0] - '0')*(b[0] - '0');  }
long int multiply(string X, string Y) {
    int n = makeEqualLength(X, Y); if (n == 0) return 0;
    if (n == 1) return multiplyiSingleBit(X, Y);
    int fh = n/2; int sh = (n-fh);
    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);
    string Yl = Y.substr(0, fh);
    string Yr = Y.substr(fh, sh);
    long int P1 = multiply(Xl, Yl);
    long int P2 = multiply(Xr, Yr);
    long int P3 = multiply(addBitStrings(Xl, Xr),
addBitStrings(Yl, Yr));
    return P1*(1<<(2*sh)) +
(P3 - P1 - P2)*(1<<sh) + P2; }
int main() { printf("%ld\n",multiply("1100","1010"));}
```

Binomial Coeficiente (C++)

```
int binomialCoeff(int n, int k) {
    int C[n+1][k+1];
    int i, j;
    for (i = 0; i <= n; i++)
        for (j = 0; j <= min(i, k); j++)
            if (j == 0 || j == i) C[i][j] = 1;
            else  C[i][j] = C[i-1][j-1] + C[i-1][j];
    return C[n][k]; }
int main() { int n = 5, k = 2;
    printf ("Value of C(%d, %d) is %d ", n, k,
binomialCoeff(n, k) ); return 0; }
```

Módulo en factorial (C++)

```
// n! % p using Wilson's Theorem
int power(int x, unsigned int y, int p) {
    int res = 1;  x = x % p;
    while (y > 0) {
        if (y & 1) res = (res*x) % p;
        y = y>>1; x = (x*x) % p; }
    return res;}
Assumption: p is prime
int modInverse(int a, int p) {
   return power(a, p-2, p); }
int modFact(int n, int p) {
    // n! % p is 0 if n >= p
    if (p <= n)   return 0;
    int res = (p-1);
    for (int i=n+1; i<p; i++)
       res  = (res * modInverse(i, p)) % p;
    return res; }
int main() {
    int n = 25, p = 29;
    cout << modFact(n, p);
    return 0; } //5
```

4 4 1 2 1 2 4 1 2 3 6 4 3 1 -> 3

Dijkstra (C++)

Pesos no negativos

```
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
struct edge{
  int from, to, weight; edge(){}
  edge(int a, int b, int c){
    from = a;   to = b; weight = c; }};
struct state{
  int node, dist; state(){}
  state(int a, int b){node = a; dist = b;  }
  bool operator<(const state &other)const{
    return other.dist < dist; } };
vector<edge> graph[MAXN];
int dist[MAXN]; int a=1, b=3; int N,E;
int dijkstra(int start, int end){  dist[start] = 0;
  priority_queue<state> pq;
  pq.push(state(start, 0));
  while(!pq.empty()){
    state cur = pq.top(); pq.pop();
    if(dist[cur.node] < cur.dist) continue;
    if(cur.node == end) return cur.dist;
    for(int i=0;i<graph[cur.node].size();i++){
      int dest = graph[cur.node][i].to;
      int wht = graph[cur.node][i].weight + cur.dist;
      if(dist[dest] <= wht) continue;
      dist[dest] = wht;
      pq.push(state(dest, wht));
    } } return -1; }
int main(){ scanf("%d %d",&N,&E);
  memset(dist,0x3f,sizeof(dist));
  for(int i=1;i<=N;i++) graph[i].clear();
       for(int i=0;i<E;i++){ int from, to, weight;
 scanf("%d %d %d",&from, &to, &weight);
    graph[from].push_back(edge(from,to,weight));
    graph[to].push_back(edge(to, from, weight));}
  printf("%d a %d %d\n",a,b,
dijkstra(a,b)); return 0; }
```

Kruskal (C++)

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
struct edge{
  int from, to, weight; edge(){}
  edge(int a, int b, int c){
    from = a;  to = b;  weight = c;}
  bool operator<(const edge &other)const{
    return weight < other.weight;} };
struct UF{
    int parents[MAXN]; int sz[MAXN];
    int components;   int mst_sum;
    UF(int n){
        for(int i=0;i<n;i++){
            parents[i] = i; sz[i] = 1;  }
        components = n; mst_sum = 0; }
    int find(int n){
        return n==parents[n] ? n : find(parents[n]);
    }
    bool isConnected(int a, int b){
        return find(a) == find(b); }
    void connect(int a, int b, int weight){
        if(isConnected(a, b)) return;
        int A,B; A = find(a); B = find(b);
        if(sz[A] > sz[B]){
            parents[B] = A; sz[A] += sz[B];  }
        else{ parents[A] = B; sz[B] += sz[A];
        } mst_sum += weight; components--; } };
int a=1; int N,E;
int main(){
        scanf("%d %d",&N,&E);
  vector<edge> edges;
  UF uf = UF(N);
        for(int i=0;i<E;i++){
    int from, to, weight; scanf("%d %d %d",&from, &to,
&weight);
    edges.push_back(edge(from,to,weight));}
  sort(edges.begin(), edges.end());
  for(int i=0;i<E;i++) uf.connect(edges[i].from,
 edges[i].to, edges[i].weight);
  printf("Kruskal de %d es %d\n",a,uf.mst_sum);
  return 0; }
```

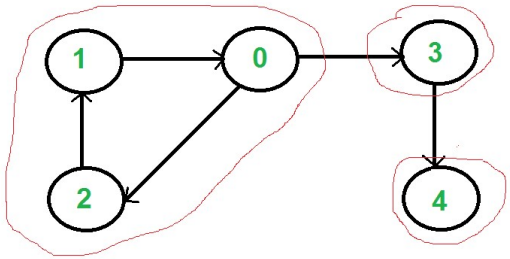
DFS (Componentes Conexas, Ciclos) y BFS (C++)

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
vector<int> graph[MAXN];
bool visited[MAXN];
int a=1, b=6; int N,E;
int DFS(int node, int target){
  if(node == target) return 0;
  if(visited[node]) return INF;
  visited[node] = true;
  int best_result = INF;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    best_result = min(
        best_result,
        DFS(dest, target)+1  );}
  return best_result; }
int BFS(int start, int target){
  queue<pair<int,int> > q;
  q.push(make_pair(start,0));
  visited[start] = true;
  while(!q.empty()){
    pair<int,int> current = q.front(); q.pop();
    if(current.first == target) return current.second;
    for(int i=0;i<graph[current.first].size();i++){
      int dest = graph[current.first][i];
      if(visited[dest]) continue;
      visited[dest] = true;
      q.push(make_pair(dest,current.second+1));
    } }
  return -1; }
```

```
int main(){
        scanf("%d %d",&N,&E);
  for(int i=1;i<=N;i++) graph[i].clear();
        for(int i=0;i<E;i++){
    int from, to; scanf("%d %d",&from, &to);
    graph[from].push_back(to);
    graph[to].push_back(from);
// borrar linea si es dirigido }
  memset(visited,0,sizeof(visited));
  printf("BFS de %d a %d es %d\n",a,b,BFS(a,b));
  memset(visited,0,sizeof(visited));
  printf("DFS de %d a %d es %d\n",a,b,DFS(a,b));
  return 0; }
// Contar ciclos
bool DFS(int node, int parent){
  if(visited[node])   return false;
  visited[node] = true;  bool res = true;
  for(unsigned int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    if(dest==parent && visited[dest])
        return true;
    if(visited[dest])  continue;
    res = DFS(dest,node); }
  return res; }

// Componentes Conexas
void dfs(int a, int b){
    visited[a][b]=true;
    for(int i=0;i<8;i++){
        int I=a+di[i];
        int J=b+dj[i];
        if(I>=0 && I<r && J>=0 && J<c &&
!visited[I][J]){
            visited[I][J]=true;
            if(arr[I][J]=='@') dfs(I,J);
        }
    }
}
void DFS(int node){
  if(visited[node]) return;
  visited[node] = true;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    DFS(dest);  } }
int main(){
  memset(visited,0,sizeof(visited));
  int comps = 0;
  for(int i=1;i<=N;i++){
    if(!visited[i]){ DFS(i); comps++; } }
  printf("Hay %d componentes\n", comps);
  return 0; }
```

Strongly Connected Components (C++)




```
/* 4 4 | 1 2 | 3 2 | 4 3 | 2 1
Output: 2 |  1 2 */
#include <bits/stdc++.h>
using namespace std;
const int MAX = 100005; int N, M;
int componentCount[2];
int componentID[2][MAX]; int degree[MAX];
bool visited[2][MAX];
vector<int> orders;
vector<int> Graph[MAX], reverseGraph[MAX];
    void dfs1 (int node) {visited[0][node] = true;
        componentID[0][node] = componentCount[0];
        for (int i : reverseGraph[node]) {
            if (!visited[0][i])  dfs1(i);  }
        orders.push_back(node); }
```

```
 void dfs2 (int node) {
     visited[1][node] = true;
     componentID[1][node] = componentCount[1];
     for (int i : Graph[node])
         if (!visited[1][i])   dfs2(i); }
 int main() {  scanf("%d%d", &N, &M);
     for (int i = 0; i < M; i ++) {int u, v;
         scanf("%d%d", &u, &v);
         Graph[u].push_back(v);
         reverseGraph[v].push_back(u); }
     for (int i = 1; i <= N; i ++) {
         if (!visited[0][i]) {
             componentCount[0] ++; dfs1(i); } }
     reverse(orders.begin(), orders.end());
     for (int i : orders) {
         if (!visited[1][i]) {
             componentCount[1] ++; dfs2(i); } }
     for (int i = 1; i <= N; i ++)
         for (int j : reverseGraph[i])
if (componentID[1][i] != componentID[1][j])
 degree[componentID[1][j]] ++; int startings = 0;
    for (int i = 1; i <= componentCount[1]; i ++)
         if (!degree[i]) startings ++;
     if (startings > 1) puts("0");
     else {  vector<int> output;
         for (int i = 1; i <= N; i ++)
             if (!degree[componentID[1][i]])
                 output.push_back(i);
         printf("%d\n", (int)output.size());
    for (int i = 0; i < (int)output.size(); i ++)
     {              printf("%d", output[i]);
    if (i < (int)output.size() - 1) putchar(' ');
   } putchar('\n');} return 0;}
```

Floyd Warshall (C++)

```
#include<bits/stdc++.h>
using namespace std; //No tiene porque ser simétrica!
#define INF 0x3F3F3F3F
   int dist[4][4] = { {  0, 5  ,INF,10},
{INF, 0  ,3  ,INF},
{INF, INF,0  ,1},
{INF, INF,INF,0}};
void floydWarshall()
{
    for (int k = 0; k < 4; k++)
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
  dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);}
int main(){ floydWarshall(); return 0;}
```

Prim (C++)

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
COPIAR EDGE Y STATE DEL DE DIJKSTRA
vector<edge> graph[MAXN];
bool visited[MAXN];
int dist[MAXN]; int a=1; int N,E;
int prim(int start){
  priority_queue<state> pq;
  pq.push(state(start, 0));  dist[start] = 0;
  int sum = 0;
  while(!pq.empty()){
    state cur = pq.top(); pq.pop();
    if(dist[cur.node] < cur.dist) continue;
    if(visited[cur.node]) continue;
    sum += cur.dist;
    visited[cur.node] = true;
    for(int i=0;i<graph[cur.node].size();i++){
      int dest = graph[cur.node][i].to;
      int wht = graph[cur.node][i].weight;
      if(visited[dest]) continue;
      pq.push(state(dest, wht));
      dist[dest] = wht; } } return sum;}
int main(){
    scanf("%d %d",&N,&E);
  memset(visited,0,sizeof(visited));
```

```
  for(int i=1;i<=N;i++) graph[i].clear(); // limpia el
grafo
    for(int i=0;i<E;i++){
    int from, to, weight; scanf("%d %d %d",&from, &to,
&weight);
    graph[from].push_back(edge(from,to,weight));
    graph[to].push_back(edge(to, from, weight)); }
  printf("Prim de %d es %d\n",a,prim(a));
  return 0;}
```

BellMan Ford (C++)

```
#include<bits/stdc++.h>
#define N 2001
#define MAX 100000000
using namespace std;
int a[N], b[N], t[N];
bool BellmanFord(int n, int m) {
    int d[N];
    fill(d, d + n, MAX);   d[0] = 0;
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < m; j++)
            if (d[a[j]] != MAX)
                if (d[a[j]] + t[j] < d[b[j]])
                    d[b[j]] = d[a[j]] + t[j];
    //negative cycle check
    for (int j = 0; j < m; j++)
        if (d[a[j]] + t[j] < d[b[j]])
            return true;  return false; }
int main() {
    int Case, n, m;
    scanf("%d", &Case);
    while (Case--)   { int i;   scanf("%d%d", &n, &m);
        for (i = 0; i < m; i++)
            scanf("%d%d%d", &a[i], &b[i], &t[i]);
puts(BellmanFord(n, m) ? "possible" : "not possible");
}  return 0; }
//Sample Input 2 - 3 3 | 0 1 1000 | 1 2 15 | 2 1 -42 |
4 4 |0 1 10 |1 2 20 |2 3 30 |3 0 -60 Sample Output
posible | not possible
```

Topological Sort (C++)

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
vector<int> graph[MAXN];
bool visited[MAXN];
stack<int> topological_order;
int N,E;
void DFS(int node){
  if(visited[node]) return;
  visited[node] = true;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i]; DFS(dest); }
  topological_order.push(node); }
int main(){
  freopen("topological.in","r",stdin);
        scanf("%d %d",&N,&E);
  for(int i=1;i<=N;i++) graph[i].clear();
        for(int i=0;i<E;i++){
    int from, to; scanf("%d %d",&from, &to);
    graph[from].push_back(to); }
  printf("Ordenamiento topologico:");
  //Asumo que el nodo 1 no es dependiente de nadie
  DFS(1);
  while(!topological_order.empty()){
    printf(" %d",topological_order.top());
    topological_order.pop(); }
  printf("\n"); return 0; }
```

Trie (C++)

```
const int put = 26; //alphabet size
struct TrieNode {
    struct TrieNode *children[put];
    //bool isEndOfWord; check end of word
    int num;};
```

```
struct TrieNode *getNode(void){
    struct TrieNode *pNode =  new TrieNode;
    //pNode->isEndOfWord = false; pNode->num=0;
for (int i = 0; i < put; i++)  pNode->children[i] =
NULL; return pNode; }
void insert(struct TrieNode *root, string key) {
    struct TrieNode *pCrawl = root;
    for (unsigned int i = 0; i < key.length(); i++) {
        int index = key[i] - 'a';
        if (!pCrawl->children[index]) pCrawl-
>children[index] = getNode();
        pCrawl->num++; //add to get repetitions
        pCrawl = pCrawl->children[index]; }
 pCrawl->num++;//pCrawl->isEndOfWord = true; }
int searchWord(char key,int a) {
   int index = key - '0';
   if (!auxNode->children[index]) return -1;
   auxNode = auxNode->children[index];
   if(auxNode->isEndOfWord) auxNode->num++;
   if (a==1) auxNode2=auxNode;
   return 0; } //Check if the word exist. Move with
pointers DONT FORGET RESTORE DE POINTER TO THE BEGIN
int search(struct TrieNode *root, string key) {
    struct TrieNode *pCrawl = root;
    for (unsigned int i = 0; i < key.length(); i++) {
        int index = key[i] - 'a';
        if (!pCrawl->children[index]) return 0;
        pCrawl = pCrawl->children[index]; }
    return pCrawl->num;}int main(){  int n,m;
    while(scanf("%d%d",&n,&m)==2){
    string word; struct TrieNode *root = getNode();
  for(int i=0; i<n;i++) {cin>>word; insert(root,word);}
    for(int i=0; i<m;i++) {cin>>word;
cout<<search(root,word)<<'\n';} } return 0;}
```

Next Permutation (C++)

```
int main() { int casos; scanf("%d", &casos);
    while (casos--) {
        int C, V, A[16] = {};
        char s[16], mm[3] = "CV";
        scanf("%d %d", &C, &V);
        for (int i = C; i < C+V; i++)
            A[i] = 1; int f = 0;
        do { for (int i = 0; i < C+V; i++)
                s[i] = mm[A[i]];
            s[C+V] = '\0';
            if (f)     putchar(' ');
            printf("%s", s), f = 1;
        } while (next_permutation(A, A+C+V));
        puts(""); }return 0; }
```

Árboles (C++)

```
int sol,pos; string aux; int solve() { pos+=2;
   if(aux[pos]=='0') return 0;
   else if(aux[pos]=='1'){
       solve(); return 0; }
   else{
       int hd = solve(); int hi = solve();
       int hijos=min(hd,hi)+1;
       sol=max(sol,hijos);return hijos; }}
int main(){
    int cases;cin>>cases;cin.ignore();
    while(cases--){  getline(cin,aux);
        sol=0,pos=-2; solve();
        cout<<sol<<endl;} return 0;}
string aux; int pos,sol;
int solve() { pos++;if(aux[pos]=='*'){
  int res = max(solve(),solve())+1;return res;}
    else return 0; }
int main() {
    int c; cin>>c; cin.ignore();
while(c--) {
        getline(cin,aux); pos=-1;  sol=0;
        printf("%d\n",solve());}
    return 0; }
```

```
***..*..*..                    3
2 2 0 1 2 0 0 2 0 0            2
```

Longest Increasing Subsequence (C++)

```
//  INPUT: a vector of integers X= 0, 8, 4, 12, 2, 10, 6 || LCS= 0,8,12
//OUTPUT: a vector containing the longest increasing subsequence
int Ceil(vector<int> &v, int l, int r, int key) {
while (r-l > 1) { int m = l + (r-l)/2;
if (v[m] >= key) r = m;
else l = m; } return r; } int LIS(vector<int> &v) {
if (v.size() == 0)   return 0;
vector<int> tail(v.size(), 0);
int length = 1;  tail[0] = v[0];
for (size_t i = 1; i < v.size(); i++) {
if (v[i] < tail[0]) tail[0] = v[i];
else if (v[i] > tail[length-1])tail[length++]=v[i];
else tail[Ceil(tail, -1, length-1, v[i])] = v[i];
  } return length;}
```

Knuth-Morris-Pratt (C++)

```
void kmp(const string &needle,const string &haystack){
int m = needle.size();
  vector<int> border(m + 1); border[0] = -1;
  for (int i = 0; i < m; ++i) {
    border[i+1] = border[i];
while (border[i+1]>-1 and needle[border[i+1]] !=
needle[i]) { border[i+1] = border[border[i+1]]; }
    border[i+1]++; }
  int n = haystack.size(); int seen = 0;
  for (int i = 0; i < n; ++i){
 while (seen > -1 and needle[seen] != haystack[i]) {
      seen = border[seen]; }
    if (++seen == m) { printf("%d\n", i - m + 1);
    seen = border[m];  } } }
int main(){  int m; bool first = true;
  while (scanf("%d",&m)==1) {
    if (!first) puts("");  first = false;
    string needle; getline(cin, needle);
    getline(cin, needle);
    string haystack; getline(cin, haystack);
    kmp(needle, haystack); } return 0;}
```

Búsqueda binaria (C++)

```
int first = 0, middle, last = arraySize - 1;
    while (first<=last) { middle = (first + last) / 2;
if (searched == array[middle])  return array[middle];
else {
if (array[middle] > searched) last = middle - 1;
else      first = middle + 1;
}    }
```

Números Primos (C++)

```
void Sieve(int n){
    bool prime[n+1];
    memset(prime, true, sizeof(prime));
    for (int p=2; p*p<=n; p++)
        if (prime[p] == true)
        for (int i=p*2; i<=n; i += p) prime[i]=false;
    for (int p=2; p<=n; p++)
       if (prime[p]) cout << p << " "; }
int main(){
    int n = 30;  Sieve(n); return 0;}
```

Constantes (C++)

PI: 4 * atan(1)
Distancia Euclediana: sqrt(pow(q[i].x-actual.x,2.0)+

pow(q[i].y-actual.y,2.0)+ pow(q[i].z-actual.z,2.0));

Distancia Manhattan: abs(x-x)+abs(y-y)

INF: 0x3F3F3F3F

```
//number too large. use powl instead of pow.
powl(a, b)
int sx[]={1,-1,0,0,1,-1,-1,1}; //8 directions
int sy[]={0,0,1,-1,1,-1,1,-1};
int sx[]={1,-1,0,0}; //4 directions
int sy[]={0,0,1,-1};
```

```
int dx2[]={-2,-1,1,2,2,1,-1,-2}; //horse jumps
int dy2[]={1,2,2,1,-1,-2,-2,-1};
scanf("%d:%d", &hr, &temp, &min);
printf("%02d:%02d\n", hr, min);
 (int)round(p, (1.0/n))
printf("%.1f\n", (a * b)/2);
void copy(first, last, result);
void swap(a,b);
void reverse(first, last);
void reverse_copy(first, last, result);
int find(const string &s2, int pos1 = 0);
cout << s1.str() << endl;
     bool operator<(const cosa &other)const{ return
     weight < other.weight; }
     pq.push(cosa{numer,2,numer});
//freopen("in.txt","r",stdin);
//freopen("out.txt","r",stdout);
```

Partitions Integer (C++)

```
//4 = 3 + 1 , 2 + 2 , 2 + 1 +1 ,1 + 1 +1 + 1
    void printAllUniqueParts(int n) {
    int p[n]; int k = 0;  p[k] = n;
    while (true) {
        printArray(p, k+1); int rem_val = 0;
        while (k >= 0 && p[k] == 1) {
            rem_val += p[k];  k--; }
        if (k < 0)  return;
        p[k]--; rem_val++;
        while (rem_val > p[k]) {
            p[k+1] = p[k];
            rem_val = rem_val - p[k]; k++;  }
        p[k+1] = rem_val; k++; } }
```

```
void bipartite(int s) { queue<int> Q;
color[s]=1; Q.push(s);
visitado[s]=true;
while(!Q.empty()&&posible) { s=Q.front();
    Q.pop(); for(unsigned int
i=0;i<adj[s].size()&&posible;++i) {
int u=adj[s][i]; if(color[u]==-1){
color[u]=1-color[s];Q.push(u);visitado[u]=true;}
else if(color[u]==color[s])posible = false; }}}
```

Longest Common Subsequence(C++)

```
string X,Y;
int memo[1005][1005];
bool mark[1005][1005];
int lcs(int m, int n ){
   if (m == 0 || n == 0) return 0;
   int &best= memo[m][n];
   if(mark[m][n]) return best;
   mark[m][n]=true;
   if (X[m-1] == Y[n-1])return best=1+lcs(m-1, n-1);
   return best=max(lcs(m, n-1),lcs(m-1, n));}
int main() {
  while(getline(cin,X)) {
      memset(mark,false,sizeof(mark));
      memset(memo,0,sizeof(memo));
  getline(cin,Y);
  printf("%d\n", lcs(X.length(),Y.length())); }
  return 0; }
```

Longest Increasing Common Subsequence (C++)

```
    // 2 3 1 6 5 4 6 AND 1 3 5 6 the LCIS is 3 5 6.
int Ceil(vector<int> &v, int l, int r, int key) {
while (r-l > 1) { int m = l + (r-l)/2;
if (v[m] >= key) r = m;
else l = m; } return r; }
int LIS(vector<int> &v) {
if (v.size() == 0)   return 0;
vector<int> tail(v.size(), 0);
int length = 1;
tail[0] = v[0];
for (size_t i = 1; i < v.size(); i++) {
if (v[i] < tail[0]) tail[0] = v[i];
else if (v[i] > tail[length-1])tail[length++]=v[i];
else tail[Ceil(tail, -1, length-1, v[i])] = v[i];
  } return length;}
```