Contents

1 String/KMP.cpp

```
O algoritmo KMP faz a busca de todas as ocorrncias de um padro em uma string.
  o algoritmo mais eficiente para matching O(n).

vector<int> f; //Vetor de falha
void prefixFunction(string P) {
        f.resize(P.size());
        f[0] = 0;
        int m = P.size();
        int j = 0;
        int i = 1;
        while (i < m) {
                if (P[j] == P[i]) {
                        f[i] = j + 1;
                        i++;
                        j++;
                }
                else if (j > 0)
                        j = f[j - 1];
                else{
                        f[i] = 0;
                        i++;
                }
        }
}
//BUSCA SUBSTRING - O(n)
int KMPmatch(string P, string T)/*(Padro, texto)*/{
        int n = T.size();
        int m = P.size();
        prefixFunction(P);
        int i = 0;
        int j = 0;
        while (i < n) {
                if (P[j] == T[i]){
                        if (j == m - 1)
                                cout<<"found at: "<<i-m+1<<endl;
//                              return i - m + 1;
                        i++;
                        j++;
                }
                else if (j > 0){
                        j = f[j - 1];
                }
                else{
                        i++;
                }
        }
        return -1;
}
```

2 String/SmallAlgorithms.cpp

```
Algoritmos teis para classificaao de strings:
        (1) isAnagram
        (2) isPalindrome

(1)
int isAnagram(string str1, string str2){
    int num1[26] = {0}, num2[26] = {0}, i = 0;

    while (str1[i] != '\0'){
        num1[str1[i] - 'a']++;
        i++;
    }
    i = 0;
    while (str2[i] != '\0'){
        num2[str2[i] -'a']++;
        i++;
    }
    for (i = 0; i < 26; i++){
        if (num1[i] != num2[i])
            return 0;
    }
    return 1;
}

(2)
int isPalindrome(string str){

        for(int i = 0; i < str.size()/2; i++){
                if(str[i] != str[str.size() - 1 - i])
                        return 0;
        }
        return 1;
}
```

3 String/LCS.cpp

O algoritmo LCS encontra a maior subsequncia comum entre duas strings, sendo uma subsequncia um conjunto de caracteres que aparecem em uma ordem de esquerda para direita, sendo estes caracteres no necessariamente consecutivos.

```
int lcs( string X, string Y, int sizeX, int sizeY)
{
   int L[sizeX+1][sizeY];
   for (int i=0; i<=sizeX; i++)
   {
     for (int j=0; j<=sizeY; j++)
     {
       if (i == 0 || j == 0)
         L[i][j] = 0;
       else if (X[i-1] == Y[j-1])
         L[i][j] = L[i-1][j-1] + 1;
       else
         L[i][j] = max(L[i-1][j], L[i][j-1]);
     }
   }
   int index = L[sizeX][sizeY];
   return index;
    // Following code is used to print LCS
   /*char lcs[index+1];
   lcs[index] = '\0';

   int i = sizeX, j = sizeY;
   while (i > 0 && j > 0)
   {
      if (X[i-1] == Y[j-1])
      {
          lcs[index-1] = X[i-1];
          i--; j--; index--;
      }

      else if (L[i-1][j] > L[i][j-1])
          i--;
      else
          j--;
   }
   return string(lcs);*/
}
```

4 String/LongestPalindrome.cpp

```
Dada uma string, o algoritmo Longest Palindrome, encontra qual o maior
palndromo na mesma. Um palndromo pode ser composto por apenas um caracter,
ou seja, toda string com ao menos um caracter possu um palndromo.
Complexidade: O(n^2)


string longestPalindromeDP(string s){
  int n = s.length();
  int longestBegin = 0;
  int maxLen = 1;
  bool table[1000][1000] = {false};
  for (int i = 0; i < n; i++) {
    table[i][i] = true;
  }
  for (int i = 0; i < n-1; i++) {
    if (s[i] == s[i+1]) {
      table[i][i+1] = true;
      longestBegin = i;
      maxLen = 2;
    }
  }
  for (int len = 3; len <= n; len++) {
    for (int i = 0; i < n-len+1; i++) {
      int j = i+len-1;
      if (s[i] == s[j] && table[i+1][j-1]) {
        table[i][j] = true;
        longestBegin = i;
        maxLen = len;
      }
    }
  }
  return s.substr(longestBegin, maxLen);
}
```

5 IncludesSTL/Vector.cpp

```
(1) Construtores
(2) Mtodos

(1) vector() // Cria um vector sem espaos alocados
    vector(int size) // Atribui um tamanho inicial ao vector
    vector<tipo>(int size, tipo contedo) /* Atribui um tamanho ao vector e
        preenche
                                            cada posi o com o contedo */
    vector(vector V) // Cria uma cpia de outro vector

(2) begin() // Retorna o endereo referente a primeira posi o do vector
    clear() // Remove todos os elementos do vector e altera seu tamanho para 0
    empty() // Verifica se o tamanho do vector   igual a 0 (retorna true caso
        seja)
    end() // Retorna um iterador referente   posi o aps o ltimo elemento
    erase(iterator) // Remove um elemento do Vector, reduzindo o tamanho do
        mesmo
    erase(iterator ini, iterator end) // Remove uma sequncia de de elementos(
        ini at  end)
    insert(iterator pos, tipo elemento) /* Insere o elemento uma posi o antes
        de pos
    pop_back() // Remove o  ltimo elemento
    pop_front() // Remove o primeiro elemento
    push_back(tipo Valor) /* Adiciona uma posi o no final do vector e preenche-
        a
                              com Valor */
    resize(int size) // Altera o tamanho do vector para size
```

6 IncludesSTL/List.cpp

```
(1) Construtores
(2) Mtodos

(1) list() // Cria uma list sem espaos alocados
    list<tipo>(int size, tipo contedo) /* Atribui um tamanho ao list e preenche
                                          cada posi o com o contedo */
    list(list V) // Cria uma cpia de outro list
    list(V.begin(), V.end()) //Cria uma cpia de uma sequncia de outro list

(2) begin() // Retorna o endereo referente a primeira posi o do list
    clear() // Remove todos os elementos do vector e altera seu tamanho para 0
    empty() // Verifica se o tamanho do list   igual a 0 (retorna true caso seja
        )
    end() // Retorna um iterador referente   posi o aps o ltimo elemento
    erase(iterator) // Remove um elemento do list, reduzindo o tamanho do mesmo
    erase(iterator ini, iterator end) // Remove uma sequncia de de elementos(
        ini at  end)
    insert(iterator pos, tipo elemento) // Insere o elemento uma posi o antes
        de pos
    pop_back() // Remove o ltimo elemento
    pop_front() // Remove o primeiro elemento
    push_back(tipo Valor) /* Adiciona uma posi o no final do list e preenche-a
                             com Valor */
    resize(int size) // Altera o tamanho do list para size
```

7 Matemtica/fatoracaoPrima.cpp

```
Retorna uma lista com os fatores primos de n.
n = P1*P2*P3*P4, sendo Pi primos e n o necessariamente distintos.

list<int> fatora(int n) {
        list<int> l;

    while (n%2 == 0) {
        l.push_back(2);
        n /= 2;
    }

    for (int i = 3; i*i <= n; i = i+2)
        while (n%i == 0) {
            l.push_back(i);
            n /= i;
        }

    if (n > 2)
        l.push_back(n);

        return l;
}
```

8 Matemtica/ud$_e xponenciacaoMatriz.cpp$

9 Matemtica/sassenfeldGaussSeidel.cpp

```cpp
bool sassenfeld(int n, vector< vector<double> > A) //garante que gaussSeidel vai
     funcionar
{
    vector<double> B(n, 1);
    for (int i = 0; i < n; i++)
    {
        double soma = 0;
        for (int j = 0; j < n; j++)
            if (j != i)
                soma += fabs(A[i][j])*B[j];
        B[i] = soma / A[i][i];
        if (B[i] >= 1)
            return 0;
    }
    return 1;
}

vector<double> gaussSeidel(int n, vector< vector<double> > A, vector<double> b,
    double Toler, int IterMax)
{
    vector<double> x(n);
    vector<double> v(n);
    for (int i = 0; i < n; i++)
    {
        double r = 1 / A[i][i];
        for (int j = 0; j < n; j++)
            if (i != j)
                A[i][j] *= r;
        b[i] *= r;
        x[i] = b[i];
    }
    int Iter = 0;
    double DifMax;
    do
    {
        Iter++;
        for (int i = 0; i < n; i++)
        {
            double Soma = 0;
            for (int j = 0; j < n; j++)
                if (i != j)
                    Soma += A[i][j] * x[j];
            v[i] = x[i];
            x[i] = b[i] - Soma;
        }
        double Norma1 = 0;
        double Norma2 = 0;
        for (int i = 0; i < n; i++)
        {
            if (abs(x[i] - v[i]) > Norma1)
                Norma1 = abs(x[i] - v[i]);
            if (abs(x[i]) > Norma2)
                Norma2 = abs(x[i]);
        }
        DifMax = Norma1 / Norma2;
```

```
    }while (DifMax >= Toler && Iter < IterMax);
    bool Erro = (DifMax >= Toler);
    return x;
}
```

10 Matemtica/catalan.cpp

```
Retorna o catalan number Cn, sequncia com a seguinte fun o:
        C0 = 1
                            n
        Cn+1 =    som Ci*Cn-i
                         i = 0

// T: O(n)
int catalan(int n) {
    return binomial(2*n, n)/(n+1);
}
```

11 Matemtica/polinomioHorner.cpp

```
Sendo f pertencente a Pn, calcula f(x).

// T: O(n), M: O(1)
int horner(int poly[], int n, int x) {
    int result = poly[0];  // Initialize result

    // Evaluate value of polynomial using Horner's method
    for (int i=1; i<n; i++)
        result = result*x + poly[i];

    return result;
}
```

12 Matemtica/phi.cpp

```
Quantidade existente de inteiros k tal que:
        k < n;
        k e n s o coprimos => mdc(k, n) = 1, mmc(k, n) = k*n;

int phi(int n) {
    int result = n;

    for (int p=2; p*p<=n; ++p) {
        if (n % p == 0) {
            while (n % p == 0)
                n /= p;
            result -= result / p;
        }
    }

    if (n > 1)
        result -= result / n;
    return result;
}
```

13 Matemtica/interpolacaoNewton.cpp

```cpp
double interpolacaoNewton(int m, vector<double> x, vector<double> y, double z)
{
    vector<double> Dely = y;
    for (int k = 1; k <= m - 1; k++)
        for (int i = m; i >= k+1; i--)
            Dely[i-1] = (Dely[i-1] - Dely[i-2]) / (x[i-1] - x[i-k-1]);
    double r = Dely[m-1];
    for (int i = m - 1; i >= 1; i--)
        r = r*(z - x[i-1]) + Dely[i-1];
    return r;
}
```

14 Matemtica/coeficienteBinomial.cpp

```
Retorna o coeficiente binomial(n, k) = n! / (k!(n-k)!)
Tambm chamado de combinatrio.

//T: O(k), M: O(1)
int binomial(int n, int k) {
    int res = 1;

    if ( k > n - k )
        k = n - k;

    for (int i = 0; i < k; ++i) {
        res *= (n - i);
        res /= (i + 1);
    }

    return res;
}
```

15 Matemtica/crivoEratostenes.cpp

```
Para todo inteiro k <= N, decide se k   primo.

vector<bool> isPrime;

// T: O(N), M: theta(N)
void crivoErathostenes(int N) {
      int rN = (int)sqrt((double)N);
      isPrime.clear();
      isPrime.resize(N + 1, true);

      for (int m = 2; m <= rN; m++)
            if (isPrime[m])
                  for (int k = m * m; k <= N; k += m)
                        isPrime[k] = false;
}
```

16 Matemtica/ehFibonacci.cpp

```
Nao funciona para numeros muito grandes, visto que n^2 estoura a capacidade do
    double.

bool isPerfectSquare(int x) {
    double s = sqrt(x);
    return (s == floor(s));
}

// T: O(logn), M: O(1)
bool isFibonacci(double n) {
    return isPerfectSquare(5*n*n + 4) ||
           isPerfectSquare(5*n*n - 4);
}
```

17 Matemtica/paradoxoAniversario.cpp

```
Dada uma sala com n pessoas onde a probabilidade de que no mnimo duas delas
    tenham o mesmo aniversrio   p, retorna n.

int find(double p) {
    return ceil(sqrt(2*365*log(1/(1-p))));
}
```

18 Matemtica/mdc.cpp

```
Computa o maior divisor comum entre dois inteiros a e b, usando o algoritmo de
    Euclides.

int mdc(int a, int b) {
        if(b == 0)
                return a;
        return mdc(b, a%b);
}
```

19 Grafos/fordFulkerson.cpp

```cpp
#include <iostream>
#include <string>
#include <string.h>
#include <iomanip>
#include <cfloat>
#include <climits>
#include <vector>
#include <set>
#include <queue>
#include <stack>
#include <map>
#include <list>
#include <algorithm>

using namespace std;

typedef int Vertice;
class Aresta {
public:
    Vertice v;
    int p;
    Aresta(Vertice v, int p) : v(v), p(p) {}
};
class ArestaOP {
public:
    Vertice o, p;
    int w;
    ArestaOP(Vertice o, Vertice p, int w) : o(o), p(p), w(w) {}
};


// MATRIZ DE ADJ PORRA
vector<vector<int> > g;
vector<vector<int> > rg; //digrafo residual
vector<int> pai;

bool bfs(int s, int t) {
        vector<bool> vis(g.size(), false);
    queue <int> q;

    q.push(s);
    vis[s] = true;
    pai[s] = -1;

    while (!q.empty()) {
        Vertice u = q.front();
        q.pop();

        for (Vertice v = 0; v < g.size(); v++)
            if (!vis[v] && rg[u][v] > 0) {
                if(v == t)
                        return true;

                q.push(v);
                pai[v] = u;
```

```
                vis[v] = true;
            }
        }

        return vis[t];
    }

    //T: O(EV^3)
    int fordFulkerson(int s, int t) {
        rg = g;
        pai.resize(g.size());

        int max_flow = 0;

        while (bfs(s, t)) {
            int path_flow = INT_MAX;

            for (Vertice v = t; v != s; v = pai[v]) {
                Vertice u = pai[v];
                path_flow = min(path_flow, rg[u][v]);
            }

            for (Vertice v = t; v != s; v = pai[v]) {
                Vertice u = pai[v];
                rg[u][v] -= path_flow;
                rg[v][u] += path_flow;
            }

            max_flow += path_flow;
        }

        return max_flow;
    }
```

20 Grafos/GrafoCiclo.cpp

```
vector<int> lbl;
vector<int> pai;
vector<vector<Vertice> > g;

bool temCicloComp(Vertice u) {
    lbl[u] = 1;

    for (int i = 0; i < g[u].size(); i++)
        if (g[u][i] != pai[u]) {
                if(lbl[g[u][i]] == 1)
                return true;
                        if(lbl[g[u][i]] == 0) {
                                pai[g[u][i]] = u;
                                if(temCicloComp(g[u][i]))
                                        return true;
                        }
        }

    lbl[u] = 2;
    return false;
}

//T: O(V + E), M: theta(V + E)
bool temCiclo() {
    lbl.clear();
    lbl.resize(g.size(), 0);
    pai.clear();
    pai.resize(g.size(), -1);

    for(Vertice u = 0; u < g.size(); u++)
        if(lbl[u] == 0) {
                pai[u] = u;
                if(temCicloComp(u))
                return true;
        }

    return false;
}
```

21 Grafos/DeteccaodePontesnoGrafo.cpp

```
vector<vector<Vertice> > g;
vector<int> pre, pai;
int qtdePontes, contaPre;

int dfs(Vertice v) {
    int minPre = pre[v] = contaPre++;

    for (int i = 0; i < g[v].size(); i++) {
        Vertice w = g[v][i];

        if (pre[w] == -1) {
            pai[w] = v;
            minPre = min(minPre, dfs(w));
        }
        else if (pai[v] != w)
            minPre = min(minPre, pre[w]);
    }

    if (minPre == pre[v])
        qtdePontes++;

    return minPre;
}

// T: O(V + E), M: theta(V)
int grafoPontes() {
        pre.clear();
        pre.resize(g.size(), -1);
        pai.clear();
    pai.resize(g.size(), -1);

        qtdePontes  = 0;
    for(Vertice u = 0; u < g.size(); u++)
        if(pre[u] == -1) {
                contaPre = 0;
                dfs(u);
        }

    return qtdePontes;
}
```

22 Grafos/minPathDAG.cpp

```
vector<vector<Aresta> > g;
vector<Vertice> tpl;
vector<bool> vis;

void dfs(Vertice u) {
        vis[u] = true;

        for(int i = 0; i < g[u].size(); i++)
                if(!vis[g[u][i].v])
                        dfs(g[u][i].v);

        tpl.push_back(u);
}

//T: O(V + E), M: theta(V)
vector<int> minPathDAG(Vertice r) {
        tpl.clear();
        tpl.reserve(g.size());
        vis.clear();
        vis.resize(g.size(), false);

        dfs(r);
        vector<int> dist(g.size(), INT_MAX);

        dist[r] = 0;
    for (int i = tpl.size() - 1; i >= 0; i--) {
        Vertice u = tpl[i];

        if(dist[u] != INT_MAX)
                for (int j = 0; j < g[u].size(); j++) {
                        Vertice w = g[u][j].v;
                        dist[w] = min(dist[w], dist[u] + g[u][j].p);
                }
    }
}
```

23 Grafos/prim.cpp

```cpp
//T: O(ElogV), M: (E + V)
int prim(vector<vector<Aresta> > &g) {
        vector<bool> pego(g.size(), false);
        multiset<ArestaOP> cand;

        pego[0] = true;
        for(int i = 0; i < g[0].size(); i++)
                cand.insert(ArestaOP(0, g[0][i]));

        int peso = 0;
        for (int i = 0; i < g.size() - 1; i++) {
                ArestaOP a;
                do {
                        if(cand.empty()) return -1; // desconexo

                        multiset<ArestaOP>::iterator it = cand.begin();
                        a = *it;
                        cand.erase(it);
                } while (pego[a.p]);

                peso += a.w;

                Vertice u = a.p;
                pego[u] = true;
                for(int i = 0; i < g[u].size(); i++)
                        cand.insert(ArestaOP(u, g[u][i]));
        }

        return peso;
}
```

24 Grafos/ComponentesConexosBFS.cpp

```
set<Vertice> //A
bfs(vector<vector<Vertice> > &g, vector<bool> &vis, Vertice r) {
        set<Vertice> ver; //A
        ver.insert(r); //A

        queue<Vertice> f;
        f.push(r);
        vis[r] = true;

        while(!f.empty()) {
                Vertice u = f.front();
                f.pop();

                for(int i = 0; i < g[u].size(); i++) {
                        Vertice w = g[u][i];

                        if(!vis[w]) {
                                ver.insert(w); //A

                                f.push(w);
                                vis[w] = true;
                        }
                }
        }

        return ver; //A
}

// T: theta(V + E), M: theta(V)
//(B) T: O(VlogV + E), M: theta(V)
int //B
componentesConexos(vector<vector<Vertice> > &g) {
    vector<bool> vis(g.size(), false);
    list<set<Vertice> > comp; //vrtices em cada componente (A)
    int c = 0; //qtd componentes (B)

    for(Vertice u = 0; u < g.size(); u++)
        if(!vis[u]) {
                c++; //B
                comp.push_back( //A
                        bfs(g, vis, u)
                        ); //A
        }

        return c; //B
}
```

25 Grafos/floydWarshall.cpp

```
// T: theta(V^3), M: theta(V^2)
vector<vector<int> > floydWarshall(vector<vector<int> > d) { //!! matriz de
    adjacencia
        int n = d.size();
        int k, i, j;

        for (k = 0; k < n; k++)
            for (i = 0; i < n; i++)
                for (j = 0; j < n; j++)
                    d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

    return d;
}
```

26 Grafos/ud$_k ruskall.cpp$

27 Grafos/GrafoBipartido.cpp

```
vector<vector<Vertice> > g;
vector<int> cor;

int dfsCor(Vertice v, int c) {
   cor[v] = 1-c;

   for (int i = 0; i < g[v].size(); i++) {
      Vertex w = g[v][i];
      if (cor[w] == -1) {
         if (dfsCor(w, 1-c) == 0) return 0;
      }
      else if (cor[w] == 1-c) return 0;
   }
   return 1;
}

int bipartido() {
        cor.clear();
        cor.resize(g.size(), -1);

        for(Vertice u = 0; u < g.size(); u++)
                if(cor[u] == -1)
                        if(dfsCor(u, 0) == 0)
                                return 0;

        return 1;
}
```

28 Grafos/dijkstraModificado.cpp

```
class HeapCS { // heap com counting sort (linear)
public:
    HeapCS(int pMin, int pMax) : pMin(pMin), pMax(pMax) {
        currentList = size = 0;
        lists = new list<Vertice>[pMax - pMin + 1];
    }

    ~HeapCS() {
        delete[] lists;
    }

    void enqueue(Vertice v, int p) {
        lists[(p - pMin)&(pMax - pMin)].push_back(v);
        size++;
    }

    Vertice dequeue() {
        while(lists[currentList].empty()) {
            currentList++;
            if(currentList == pMax - pMin + 1) currentList = 0;
        }

        Vertice v = lists[currentList].back();
        lists[currentList].pop_back();
        size--;

        return v;
    }

    int pMin, pMax;
    list<Vertice> *lists;
    int currentList, size;
};

vector<int> d; //distancias

//T: O(V + E), M: theta(V + E)
void dijkstra(vector<vector<Aresta> > &g, int r, int pMin, int pMax) {
    int n = g.size();

        HeapCS Q(pMin, pMax);
        d.clear();
        d.resize(n, INT_MAX);
    vector<bool> cor(n, false);

    d[r] = 0;
    cor[r] = true;

        int u = r;
        for (int i = 0; i < g[u].size(); i++) {
        if (d[u] + g[u][i].p < d[g[u][i].v])
            d[g[u][i].v] = d[u] + g[u][i].p; //relaxamento
        if (!cor[g[u][i].v]) {
            cor[g[u][i].v] = true;
            Q.enqueue(g[u][i].v, g[u][i].p);
```

```
        }
    }

    while (Q.size != 0) {
        int u = Q.dequeue();
        //if(procurado == u) return;

        for (int i = 0; i < g[u].size(); i++) {
            if (d[u] + g[u][i].p < d[g[u][i].v])
                d[g[u][i].v] = d[u] + g[u][i].p; //relaxamento
            if (!cor[g[u][i].v]) {
                cor[g[u][i].v] = true;
                Q.enqueue(g[u][i].v, g[u][i].p);
            }
        }
    }
}
```

29 Grafos/dijkstra.cpp

```
vector<int> d; //distancias

bool comp(const int A, const int B) {
    return d[A] > d[B];
}

//T: O(ElogV), M: theta(V + E)
void dijkstra(vector<vector<Aresta> > &g, int r) {
    int n = g.size();

        vector<Vertice> Q;
        d.clear();
        d.resize(n, INT_MAX);
    vector<bool> cor(n, false);

    Q.push_back(r);
    d[r] = 0;
    cor[r] = true;

    while (!Q.empty()) {
        int u = Q[0];
        //if(procurado == u) return;
        pop_heap(Q.begin(), Q.end(), comp);
                Q.pop_back();

        for (int i = 0; i < g[u].size(); i++) {
            if (d[u] + g[u][i].p < d[g[u][i].v])
                d[g[u][i].v] = d[u] + g[u][i].p; //relaxamento
            if (!cor[g[u][i].v]) {
                cor[g[u][i].v] = true;
                Q.push_back(g[u][i].v);
            }
        }

        make_heap(Q.begin(), Q.end(), comp);
    }
}
```

30 Grafos/ud$_b$oruvka.cpp

31 Grafos/digrafoCiclo.cpp

```
vector<int> lbl;
vector<vector<Vertice> > g;

bool temCicloComp(Vertice u) {
    lbl[u] = 1;

    for (int i = 0; i < g[u].size(); i++)
        if (lbl[g[u][i]] == 1 || (lbl[g[u][i]] == 0 && temCicloComp(g[u][i])) )
            return true;

    lbl[u] = 2;
    return false;
}

//T: O(V + E), M: theta(V + E)
bool temCiclo() {
        lbl.clear();
        lbl.resize(g.size(), 0);

        for(Vertice u = 0; u < g.size(); u++)
                if(lbl[u] == 0)
                        if(temCicloComp(u))
                                return true;

        return false;
}
```

32 Grafos/ud$_johnson.cpp$

33 Grafos/bellmanFord.cpp

```cpp
vector<Vertice> o, d; //origem e destino da aresta i
vector<int> p; //peso da aresta i
vector<int> dist;
int V, E; //numero vertices e arestas

//T: O(VE), M: O(V)
bool bellmanFord(Vertice r) {
        dist.clear();
        dist.resize(V, INT_MAX);
    dist[r] = 0;

    for (int i = 1; i <= V-1; i++) {
        for (int j = 0; j < E; j++) {
            Vertice u = o[j], v = d[j];
            int w = p[j];
            if (dist[u] != INT_MAX && dist[u] + w < dist[v])
                dist[v] = dist[u] + w;
        }
    }

    for (int j = 0; j < E; j++) {
        Vertice u = o[j], v = d[j];
        int w = p[j];
        if (dist[u] != INT_MAX && dist[u] + w < dist[v])
            return 0; //contem ciclo negativo, informacoes invalidas
    }

    return 1;
}
```

34 Grafos/BFS.cpp

```cpp
// T: theta(V + E), M: theta(V)
void bfs(vector<vector<Vertice> > &g, Vertice r) {
        vector<bool> vis(g.size(), false);
        vector<Vertice> pai(g.size(), -1); //pais, arvore de busca (A)
        vector<int> dis(g.size(), INT_MAX); //distancia ate a raiz (B)

        queue<Vertice> f;
        f.push(r);
        dis[r] = 0; //B
        vis[r] = true;

        while(!f.empty()) {
                Vertice u = f.front();
                f.pop();

                for(int i = 0; i < g[u].size(); i++) {
                        Vertice w = g[u][i];

                        if(!vis[w]) {
                                pai[w] = u; //A
                                dis[w] = dis[u] + 1; //B

                                f.push(w);
                                vis[w] = true;
                        }
                }
        }
}
```

35 Grafos/tarjan.cpp

```cpp
#include <iostream>
#include <string>
#include <string.h>
#include <iomanip>
#include <cfloat>
#include <climits>
#include <vector>
#include <set>
#include <queue>
#include <stack>
#include <map>
#include <list>
#include <algorithm>

using namespace std;

typedef int Vertice;
class Aresta {
public:
    Vertice v;
    int p;
    Aresta(Vertice v, int p) : v(v), p(p) {}
};
class ArestaOP {
public:
    Vertice o, p;
    int w;
    ArestaOP(Vertice o, Vertice p, int w) : o(o), p(p), w(w) {}
};

vector<vector<Vertice> > g;
vector<int> pre;
vector<int> low;
vector<int> sc; //componente forte de cada vertice
stack<Vertice> pil;
int kk;
int N;
int scnum;

void dfs(Vertice v) {
   pre[v] = kk++;
   low[v] = pre[v];
   pil.push(v);

   for (int i = 0; i < g[v].size(); i++) {
      Vertice w = g[v][i];
      if (pre[w] == -1)
         dfs(w);
      if (low[w] < low[v])
         low[v] = low[w];
   }

   if (low[v] < pre[v])
      return;
```

```
    do {
       Vertice u = pil.top();
           pil.pop();
       sc[u] = scnum;
       low[u] = g.size();
    } while (pil.top() != v);
    scnum++;
}

int tarjan() {
     pre.clear();
         pre.resize(g.size(), -1);
         low.clear();
         low.resize(g.size());
         sc.clear();
         sc.resize(g.size());

     scnum = 0; N = 0; kk = 0;
         for(Vertice u = 0; u < g.size(); u++)
                 if(pre[u] == -1)
                         dfs(u);
}
```

36 Grafos/kosarajuSharir.cpp

```
vector<vector<Vertice> > g;
vector<vector<Vertice> > gg; //digrafo com as arestas invertidas
vector<bool> vis;
vector<int> pos;
vector<int> comp; //componente forte de cada vertice
int qtdComps;

void dfs(vector<vector<Vertice> > &graf, Vertice v) {
    vis[v] = true;

    for(int i = 0; i < graf[v].size(); i++) {
        Vertice u = graf[v][i];
        if(!vis[u])
            dfs(graf, u);
    }

    pos.push_back(v);
    comp[v] = qtdComps;
}

//T: O(n), M: theta(n)
int kosarajuSharir() {
        vis.clear();
    vis.resize(g.size(), false);
    pos.clear();
    pos.reserve(g.size());
    comp.resize(g.size());

    for(Vertice u = 0; u < gg.size(); u++)
        if(!vis[u])
                        dfs(gg, u);

        vector<Vertice> posGG = pos;
        vis.clear();
    vis.resize(g.size(), false);
    pos.clear();

    qtdComps = 0;
        for(int i = 0; i < posGG.size(); i++) {
                Vertice u = posGG[i];
                if(!vis[u]) {
                        dfs(g, u);
                        qtdComps++;
                }
        }

        return qtdComps;
}
```

37 PD/SomatoriosdosSubconjuntos.cpp

```cpp
// T: theta(nK), M: theta(K), sendo K o maior valor de interesse
vector<bool> somatoriosDosSubconjuntos(int conj[], int n, int K) {
        vector<bool> pos(K + 1, false);
        pos[0] = true;

        for(int i = 0; i < n; i++)
                for(int j = K; j >= conj[i]; j--)
                        if(pos[j - conj[i]])
                                pos[j] = true;

        return pos;
}
```

38 PD/ud$_k$napSackLimitado.cpp

39 PD/knapSack01.cpp

```cpp
// T: theta(nW), M: theta(nW), sendo W o pesoMaximo

int knapSack01(int pesoMaximo, int peso[], int val[], int n) {
        int i, w;
        int K[n+1][pesoMaximo + 1];

        for (i = 0; i <= n; i++)
        {
        for (w = 0; w <= pesoMaximo; w++)
        {
                if (i==0 || w==0)
                        K[i][w] = 0;
            else if (peso[i-1] <= w)
                K[i][w] = max(val[i-1] + K[i-1][w-peso[i-1]],  K[i-1][w]);
            else
                K[i][w] = K[i-1][w];
        }
    }

    return K[n][pesoMaximo];
}
```

40 PD/ut$_k$napSackIlimitado.cpp

```
// T: theta(nW), M: theta(W), sendo W o pesoMaximo

int knapSackIlimitado(int pesoMaximo, int peso[], int val[], int n) {
        int i, w;
        int K[pesoMaximo + 1];

        K[0] = 0;
        for (w = 1; w <= pesoMaximo; w++) {
                K[w] = 0;

                for(int i = 0; i < n; i++)
           if(peso[i] <= w)
               K[w] = max(K[w], K[w - peso[i]] + val[i]);
        }

        return K[pesoMaximo];
}
```

41 Geometria/MinimumCostPolygonTriangulation.cpp

```
// A Dynamic Programming based program to find minimum cost of convex
// polygon triangulation

#define MAX 1000000.0


// Structure of a point in 2D plane
struct Point
{
    int x, y;
};

// Utility function to find minimum of two double values
double min(double x, double y)
{
    return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y));
}

// A utility function to find cost of a triangle. The cost is considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
    Point p1 = points[i], p2 = points[j], p3 = points[k];
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A Dynamic programming based function to find minimum cost for convex
// polygon triangulation.
double mTCDP(Point points[], int n)
{
   // There must be at least 3 points to form a triangle
   if (n < 3)
      return 0;

   // table to store results of subproblems.  table[i][j] stores cost of
   // triangulation of points from i to j.  The entry table[0][n-1] stores
   // the final result.
   double table[n][n];

   // Fill table using above recursive formula. Note that the table
   // is filled in diagonal fashion i.e., from diagonal elements to
   // table[0][n-1] which is the result.
   for (int gap = 0; gap < n; gap++)
   {
      for (int i = 0, j = gap; j < n; i++, j++)
      {
          if (j < i+2)
```

```
            table[i][j] = 0.0;
          else
          {
              table[i][j] = MAX;
              for (int k = i+1; k < j; k++)
              {
                double val = table[i][k] + table[k][j] + cost(points,i,j,k);
                if (table[i][j] > val)
                     table[i][j] = val;
              }
          }
      }
   }
   return  table[0][n-1];
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
    int n = sizeof(points)/sizeof(points[0]);
    cout << mTCDP(points, n);
    return 0;
}
```

42 Geometria/ConvexHull.cpp

```
Convex Hull   o menor polgono convexo que, dado um conjunto disperso de pontos
    P (vector<Ponto>P),
consegue abranger no seu interior todos os pontos do conjunto P com o menor
    numero de arestas


bool operator <(const Ponto &p1, const Ponto &p2) {
        return p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y);
}

// Fun o que auxilia a constru o das bordas superior e inferior.
// Verifica a "curva" da liga o entre dois pontos consecutivos em x, o retorno
// tem o seguinte comportamento:
//    - > 0 caso ocorra uma "curva" para esquerda;
//    - < 0 caso ocorra uma "curva" para a direita;
//    - == 0 caso os pontos sejam coolineares.
int verCurva(const Ponto &O, const Ponto &A, const Ponto &B)
{
        return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}

// Retorna a lista de pontos que representam o convex hull em ordem ant -horria
    .
// Obs: O ltimo ponto da lista retornada   o mesmo que o primeiro.
vector<Ponto> convex_hull(vector<Ponto> P)
{
        int n = P.size(), k = 0;
        vector<Ponto> H(2*n);

        // Ordena os pontos
        // Obs: Conforme fun o que sobrecarrega o operador "<"
        sort(P.begin(), P.end());

        // Constri a borda inferior do convex hull
        for (int i = 0; i < n; i++) {
                while (k >= 2 && verCurva(H[k-2], H[k-1], P[i]) <= 0) k--;
                H[k++] = P[i];
        }

        // Constri a borda inferior do convex hull
        for (int i = n-2, t = k+1; i >= 0; i--) {
                while (k >= t && verCurva(H[k-2], H[k-1], P[i]) <= 0) k--;
                H[k++] = P[i];
        }

        H.resize(k);
        return H;
}
```

43 Geometria/Poligonos.cpp

```
A representa o de Poligonos   a apresentada no (1), algoritmos relacionados com
     poligonos:
        (1) Representacao Poligono
        (2) Area do Poligono
        (3) Teste de pertinencia do Ponto no Poligono (isInside())


(1)
vector<Ponto> Poligono;

(2)
double areaPoligono(vector<Ponto> Poligono){

        double total = 0;

        for (int i = 0; i < Poligono.size(); i++){
        int j = (i + 1) % Poligono.size();
        total += (Poligono[i].x * Poligono[j].y) - (Poligono[j].x * Poligono[i].
            y);
    }
    return total / 2;
}

(3)
#define INF 10000
//              Basicos (5) - interPtSeg (q,p,r)
//              Intersecao(1) -  interSegSeg(p1,q1,p2,q2)


// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Ponto p, Ponto q, Ponto r){
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;  // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// Returns true if the Ponto p lies inside the Poligono
bool isInside(vector<Ponto> Poligono, Ponto p){

    if (Poligono.size() < 3)  return false;

    Ponto extreme(INF, p.y);

    int count = 0, i = 0;

        do{
        int next = (i+1)%Poligono.size();

        if (interSegSeg(Poligono[i], Poligono[next], p, extreme)) {
            if (orientation(Poligono[i], p, Poligono[next]) == 0)
               return interPtSeg(p, Poligono[i], Poligono[next]);
```

```
            count++;
        }

        i = next;
    } while (i != 0);

    return count&1;  // Same as (count%2 == 1)
}
```

44 Geometria/Basicos.cpp

```
Fun es b sicas usadas em distintos algoritmos tem a representacao de ponto e
    fun es simples:
        (1)      representacao do Ponto
        (2)  Produto Escalar
        (3)  Produto Vetorial
        (4)  Produto Vetorial (sinal)
        (5)  Teste de pertinencia de ponto em segmento
        (6)      Distancia Euclidiana
        (7)  Pontos Colineares


(1)
class Ponto{
        public:
                int x;
                int y;
                Ponto(){}
                Ponto(int x, int y):x(x), y(y){}

};


(2)
long long prodEscalar(Ponto origem, Ponto a, Ponto b) {
        long long v1 = ((long long) a.x-origem.x)*(b.x-origem.x);
        long long v2 = ((long long) a.y-origem.y)*(b.y-origem.y);

        return v1+v2;
}

(3)
long long prodVetorial(Ponto origem, Ponto a, Ponto b) {
        long long v1 = ((long long) a.x-origem.x)*(b.y-origem.y);
        long long v2 = ((long long) a.y-origem.y)*(b.x-origem.x);

        return v1-v2;
}

(4)
int prodvetsn(Ponto origem, Ponto a, Ponto b) {
        long long v1 = ((long long) a.x-origem.x)*(b.y-origem.y);
        long long v2 = ((long long) a.y-origem.y)*(b.x-origem.x);

        if( v1 < v2 )  return -1;
        else if( v1 > v2 ) return +1;
        else return 0;
}

(5)
bool interPtSeg(Ponto p, Ponto a, Ponto b) {
        return prodVetorial(p, a, b)==0 && prodEscalar(a, p, b)>=0 &&
            prodEscalar(b, p, a)>=0;
}

(6)
```

```
double distancia(Ponto a, Ponto b){
        return sqrt(pow(b.x - a.x, 2) + pow(b.y - a.y, 2));
}

(7)
bool ptsColineares(Ponto a, Ponto b, Ponto c){

        int s = (c.y - b.y) * a.x + (b.x - c.x) * a.y + (c.x * b.y - b.x * c.y);

    if (s < 0 || s > 0)
        return false;
    else
        return true;
}
```

45 Geometria/ProblemadosParesmaisProximos.cpp

```
Usando o Paradigma de Diviso e Conquista, o problema dos pares mais prximos
    retorna a menor distncia entre dois pares dado um N pontos
(1) codigo
(2) como usar o codigo (Main)


(1)
bool operator <(Ponto a, Ponto b){
    return a.x < b.x;
}

vector<Ponto> P;

// Fun o de compara o para ordenar um vetor de indices(int) a partir do eixo Y
     do vetor de pontos P.
bool compP(int a, int b){
    if(P[a].y == P[b].y)
        return a < b;
    return P[a].y < P[b].y;
}

double parDePontosMaisProximos(int a, int b, vector<int> Y){
    // Se a quantidade de pontos a serem analisados forem de at  3. (Caso Base)
        if(b-a <= 3){
        double md = DBL_MAX, d;

        for(int i = a; i < b; i++){
            for(int j = i+1; j < b; j++){
                //faz a distancia dos 3 pontos com os 3 pontos
                d = sqrt(pow(P[i].x - P[j].x, 2.0) + pow(P[i].y - P[j].y, 2.0));
                //e ja analisa qual   melhor dist
                                md = min(d, md);
            }
        }

        return d;
        //retorna a menor distancia
        }

        //conta pra dividir no eixo y
    int m = a+(b-a)/2;
    double d1, d2, md;
    vector<int> Y1(m-a), Y2(b-m);
    int i1 = 0, i2 = 0;
    for(int i = 0; i < Y.size(); i++){
        //se o indice contido em y for menor que a conta para dividir ao meio o
            conjunto de pontos, joga para o lado y1
                if(Y[i] < m){
            Y1[i1] = Y[i];
            i1++;
        }
        //se nao joga para o lado y2
        else{
            Y2[i2] = Y[i];
            i2++;
```

```
        }
    }
    //duas distancias- d1-Y1(lado esquerdo - de a ate m - exemplo: 0 a 2) e d2-
        Y2(lado direito - de m ate b - exemplo:2 a 5)
    d1 = parDePontosMaisProximos(a, m, Y1);
    d2 = parDePontosMaisProximos(m, b, Y2);
    //pega a menor distancia entre d1 e d2
        md = min(d1, d2);

    int tam = 0;
    vector<int> Yl(b-a);
     //ponto mais proximos no vetor yl
    for(int i = 0; i < Y.size(); i++){
        if(fabs(P[Y[i]].x - P[m].x) < md){
            Yl[tam] = Y[i];
            tam++;
        }

    }
    //analisa a fronteira e pontos mais proximo(menor distancia)
    for(int i = 0; i < tam; i++){
        for(int j = i+1; j-i < 8 && j < tam; j++){
            d1 = sqrt( pow(P[Yl[i]].x - P[Yl[j]].x, 2.0) + pow(P[Yl[i]].y - P[Yl
                [j]].y, 2.0) );
            md = min(md, d1);
        }
    }

    return md;
}


(2)
int main(){

           vector<int> Y;

        for(int i = 0; i < n; i++){
            cin >> P[i].x >> P[i].y;
            Y[i] = i;
        }

        sort(P.begin(), P.end());
        sort(Y.begin(), Y.end(), compP);

        d = parDePontosMaisProximos(0, n, Y);
}
```

46 Geometria/Intersecao.cpp

```
Alguns Algoritmos de interse o usam fun es de Basicos:
        (1) Intersecao entre dois segmentos
        (2) Intersecao entre dois retangulos


(1)
//interse o do segmento dos pontos a e b (AB) com o segmento c e d (CD)
bool interSegSeg(Ponto a, Ponto b, Ponto c, Ponto d) {
        int i, r1, r2;

                if( min(a.x,b.x) > max(c.x,d.x) || max(a.x,b.x) < min(c.x,d.x) )
                        return 0;
                if( min(a.y,b.y) > max(c.y,d.y) || max(a.y,b.y) < min(c.y,d.y) )
                        return 0;

                //USAR FUNCAO BASICO
        r1 = prodvetsn(a, c, b) * prodvetsn(a, d, b);
        r2 = prodvetsn(c, a, d) * prodvetsn(c, b, d);

        return r1<=0 && r2<=0;
}

(2)
bool interRetan(Ponto l1, Ponto r1, Ponto l2, Ponto r2){
    // If one rectangle is on left side of other
    if (l1.x > r2.x || l2.x > r1.x)
        return false;

    // If one rectangle is above other
    if (l1.y < r2.y || l2.y < r1.y)
        return false;

    return true;
}
```

47 Geometria/SmallestCircle.cpp

```
#include<iostream>
#include<cstdlib>
#include<math.h>
#include<cstdio>
#include<list>

using namespace std;

class Point {
        public:
           double x;
           double y;

        Point(){
        this->x=0;
        this->y=0;
    }

        Point(double x, double y) {
                this->x = x;
                this->y = y;
        }
        Point subtract(Point p) {
                Point aux;
                aux.x=this->x-p.x;
                aux.y=this->y-p.y;
                return aux;
        //return new Point(x - (p.x), y - (p.y));
        }
        double distance(Point p) {
                return hypot(x - p.x, y - p.y);;
        }
        // Signed area / determinant thing
        double cross(Point p) {
                return x * p.y - y * p.x;
        }
        // Magnitude squared
        double norm() {
                return x * x + y * y;
        }
};

class Circle {
        public:
           double static const EPSILON = 1e-12;
           Point c;   // Center
           double r;  // Radius

        Circle(){
        this->c.x=-1;
        this->c.y=-1;
                this->r = -1;
    }

        Circle(Point c, double r) {
```

```
                this->c = c;
                this->r = r;
        }
        bool contains(Point p) {
                return (c.distance(p) <= r + EPSILON);
        }
        bool contains(list<Point> ps) {
                for (list<Point>::iterator it=ps.begin(); it != ps.end(); ++it)
                    {
                        if (!contains(*it))
                                return false;
                }
                return true;
        }
};

Circle makeDiameter(Point a, Point b) {
    Circle aux;
    Point c;
    double r;
    c.x=(a.x + b.x)/ 2.0;
    c.y=(a.y + b.y) / 2.0;
    r=a.distance(b) / 2.0;
    aux.c=c;
    aux.r=r;
        return aux;
}

Circle makeCircumcircle(Point a, Point b, Point c) {
        double d = (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y)) *
             2;
        Circle aux;
        if (d == 0)
                return aux;
        aux.c.x = (a.norm() * (b.y - c.y) + b.norm() * (c.y - a.y) + c.norm() *
            (a.y - b.y)) / d;
        aux.c.y = (a.norm() * (c.x - b.x) + b.norm() * (a.x - c.x) + c.norm() *
            (b.x - a.x)) / d;
        aux.r=aux.c.distance(a);
        return aux;
}

Circle makeCircleTwoPoints(list<Point> points, Point p, Point q) {
        Circle temp = makeDiameter(p, q);
        if (temp.contains(points))
                return temp;

        Circle left;
        Circle right;
        for (list<Point>::iterator it=points.begin(); it != points.end(); ++it)
            {  // Form a circumcircle with each point
                Point pq = q.subtract(p);
                double cross = pq.cross((*it).subtract(p));
                Circle c = makeCircumcircle(p, q, (*it));
                if (c.r==-1)
                        continue;
                else if (cross > 0 && (left.r==-1 || pq.cross(c.c.subtract(p)) >
```

```
                    pq.cross(left.c.subtract(p))))
                        left = c;
                else if (cross < 0 && (right.r==-1 || pq.cross(c.c.subtract(p))
                    < pq.cross(right.c.subtract(p))))
                        right = c;
        }
        return right.r==-1 || left.r!=-1 && left.r <= right.r ? left : right;
}

Circle makeCircleOnePoint(list<Point> points, Point p) {
        Circle c; //new Circle(p, 0)
        c.c.x=p.x;
        c.c.y=p.y;
        c.r=0;
        for (list<Point>::iterator it=points.begin(); it != points.end(); ++it)
            {
                Point q = (*it);
                if (!c.contains(q)) {
                        if (c.r == 0){
                                c = makeDiameter(p, q);
            }
                        else{
                list<Point> aux;
                for(list<Point>::iterator ite=points.begin(); ite!=it; ++ite){
                    aux.push_back(*ite);
                }
                                c = makeCircleTwoPoints(aux, p, q);
            }
                }
        }
        return c;
}

Circle makeCircle(list<Point> points) {
        // Clone list to preserve the caller's data, randomize order
        list<Point> shuffled = points;
        //Collections.shuffle(shuffled, new Random());

        // Progressively add points to circle or recompute circle
        Circle c;
        for (list<Point>::iterator it=shuffled.begin(); it != shuffled.end(); ++
            it){
                Point p = (*it);
        if (c.r==0 || !c.contains(p)){
                    list<Point> aux;
                    for(list<Point>::iterator ite=shuffled.begin(); ite!=it; ++
                        ite){
                    aux.push_back(*ite);
                }
                        c = makeCircleOnePoint(aux, p);
        }
        }
        return c;
}

int main(){
```

```
    return 0;
}
```

48 utility.cpp

```
#include <iostream>
#include <string>
#include <string.h>
#include <iomanip>
#include <cfloat>
#include <climits>
#include <vector>
#include <set>
#include <queue>
#include <stack>
#include <map>
#include <list>
#include <algorithm>

using namespace std;

#define FOR(i,n) for(int i = 0; i < n; i++)
#define FORit(it,container) for(it = container.begin(); it != container.end();
    it++)

typedef int Vertice;
class Aresta {
public:
    Vertice v;
    int p;
    Aresta(Vertice v, int p) : v(v), p(p) {}
};
class ArestaOP {
public:
        Vertice o, p;
        int w;
        ArestaOP(Vertice o, Vertice p, int w) : o(o), p(p), w(w) {}
};
```