Notebook - Maratona de Programação

Tiago de Souza Fernandes

Contents

1 Algoritmos

1.1 Recursive-BS.cpp

```cpp
// Recursive binary search

int bs(int x, int ini, int fim)
{
    if(fim>=ini)
    {
        int meio = (ini+fim)/2;

        if(vetor[mid]==x)
            return x;

        if(vetor[meio]<x)
            return bs(x, ini, meio-1);
        else
            return bs(x, meio+1, fim);
    }

    return -1;
}
```

2 Grafos

2.1 BFS.cpp

```cpp
//BFS (Breadth First Search) O(V+A)

void BFS(int x)
{
    int atual, v, u;
    queue<int> fila;
    fila.push(x);

    componente[x] = valor;
    atual = 0;
    while(!fila.empty())
    {
        v = fila.front();
        fila.pop();

        for(int i = 0;i < (int)vizinhos[v].size();i
++)
        {
            u = vizinhos[v][i];
            if(componente[u] == -1)
            {
                componente[u] = componente[v];
                fila.push(u);
            }
        }
    }
}
```

2.2 Dijkstra.cpp

```cpp
// Dijkstra - Shortest Path

#define pii pair<int, int>
#define vi vector<int>
#define vii vector< pair<int,int> >
#define INF 0x3f3f3f3f

vector<vii> grafo;
vi distancia;
priority_queue< pii, vii, greater<pii> > fila;

void dijkstra(int k)
{
    int dist, vert, aux;
    distancia[k]=0;

    fila.push(mp(k, 0));

    while(!fila.empty())
    {
        aux=fila.top().f;
        fila.pop();

        for(int i=0; i<grafo[aux].size(); i++)
        {
            vert=grafo[aux][i].f;
            dist=grafo[aux][i].s;
            if(distancia[vert]>distancia[aux]+dist)
            {
                distancia[vert]=distancia[aux]+dist;
                fila.push(mp(vert, distancia[vert]));
            }
        }
    }
}

int main()
{
    dist.assign(N+1, INF);
    grafo.assign(N+1, vii());

    for(int i=0; i<M; i++)
    {
        cin >> a >> b >> p;
        grafo[a].pb(mp(b, p));
        grafo[b].pb(mp(a, p));
    }
}
```

2.3 Floyd-Warshall.cpp

```cpp
// Floyd Warshall

int dist[MAX][MAX];

void Floydwarshall()
{
    for(int k = 1;k <= n;k++)
        for(int i = 1;i <= n;i++)
            for(int j = 1;j <= n;j++)
                dist[i][j] = min(dist[i][j], dist[i][
    k] + dist[k][j]);
}
```

2.4 Kruskal.cpp

```cpp
// Kruskal -  Minimum Spanning Tree

typedef struct
{
    int A, B;
    int dist;
} vertice;

vertice grafo[MAX];
int pai[MAX];

int find(int X) // Union-Find
{
    if(pai[X]==X)
        return X;
    else
        return pai[X]=find(pai[X]);
}
```

```cpp

void join(int X, int Y)
{
    int paix = find(X);
    int paiy = find(Y);
    pai[paix]=paiy;
}

bool comp(vertice A, vertice B)
{
    return A.dist < B.dist;
}

void kruskal()
{
    for(int i=1;i<=N;i++)
        pai[i]=i;

    for(int i=1;i<=M;i++)
        cin >> grafo[i].A >> grafo[i].B >> grafo[i].
    dist;

    sort(grafo+1, grafo+M+1, comp);

    for(int i=1;i<M;i++)
    {
        if(find(grafo[i].A)!=find(grafo[i].B))
        {
            join(grafo[i].A, grafo[i].B);
            soma+=grafo[i].dist;
        }
    }

    cout << soma << endl;
}
```

2.5 DFS.cpp

```cpp
//DFS (Depth First Search) O(V+A)

void DFS(int x)
{
    for(int i=0; i<(int)vizinhos[x].size(); i++)
    {
        int v = vizinhos[x][i];
        if(componente[v] == -1)
        {
            componente[v] = componente[x];
            DFS(v);
        }
    }
}
```

2.6 Represent.cpp

```cpp
// Grafos

// List of edges

    vector< pair<int, int> > arestas;
    arestas.push_back(make_pair(1, 2));
    arestas.push_back(make_pair(1, 3));

// Adjacency Matrix

    int grafo[10][10];

    grafo[1][2] = grafo[2][1] = 1;
    grafo[1][3] = grafo[3][1] = 2;

// Adjacency List

    vector<int> vizinhos[10];

    vizinhos[1].push_back(2);
    vizinhos[1].push_back(2);
```

3 Geometria

3.1 Inter-Retas.cpp

```cpp
// Intersection between lines

typedef struct
{
    int x, y;
} pnt;

bool collinear(pnt p, pnt q, pnt r)
{
    if(q.x<=max(p.x,r.x) && q.x>=min(p.x,r.x) && q.y
    <=max(p.y,r.y) && q.y>=min(p.y,r.y))
        return true;

    return false;
}

int orientation(pnt p, pnt q, pnt r)
{
    int val=(q.y-p.y)*(r.x-q.x)-(q.x-p.x)*(r.y-q.y);

    if(val==0)
        return 0;
    else if(val>0)
        return 1;
    else
        return 2;
}

bool intersect(pnt p1, pnt q1, pnt p2, pnt q2)
{
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if(o1!=o2 and o3!=o4)
        return true;

    if(o1==0 && collinear(p1, p2, q1))
        return true;

    if(o2==0 && collinear(p1, q2, q1))
        return true;

    if(o3==0 && collinear(p2, p1, q2))
        return true;

    if(o4==0 && collinear(p2, q1, q2))
        return true;

    return false;

}
```

4 ED

4.1 Iterative-SegTree.cpp

```cpp
// Segment Tree Iterativa - Range maximum query

#define N 100010
```

```

struct Segtree
{
    int t[2*N]={0};

    void build()
    {
        for(int i=N-1; i>0; i--)
            t[i]=max(t[i<<1], t[1<<1|1]);
    }

    int query(int l, int r)
    {
        int ans=0;
        for(i+=N, r+=N; l<r; l>>=1, r>>=1)
        {
            if(l&1)
                ans=max(ans, t[l++]);
            if(r&1)
                ans=max(ans, t[--r]);
        }

        return ans;
    }

    void update(int p, int value)
    {
        for(t[p+=n]=value; p>1; p>>=1)
            t[p>>1]= max(t[p], t[p^1]);
    }

};

int main()
{
    Segtree st;

    for(int i=0;i<n;i++)
    {
        cin >> aux;
        st.t[N+i]=aux; //Leaves are stored in
    continuous nodes with indices starting with N
    }

    st.build();
    x = st.query(inicio, fim);
    st.update(ind, value);

}
```

4.2 Recursive-SegTree.cpp

```
// Segment Tree Recursiva - Range maximum query

vector<int> val(MAX, 0);
vector<int> vet(N);

void monta(int i, int j, int no)
{
    if(i==j)
    {
        val[no]=vet[i];
        return;
    }

    int esq = 2*no;
    int dir = 2*no+1;
    int meio = (i+j)/2;

    monta(i, meio, esq);
    monta(meio+1, j, dir);

    val[no]=max(val[esq], val[dir]);
}

void atualiza(int no, int i, int j, int pos, int
    novo_valor)
{
    if(i==j)
    {
        val[no]=novo_valor;
    }else
    {
        int esq = 2*no;
        int dir = 2*no+1;
        int meio = (i+j)/2;

        if(pos<=meio)
            atualiza(esq, i, meio, pos, novo_valor);
        else
            atualiza(dir, meio+1, j, pos, novo_valor)
    ;

        if(val[esq]>val[dir])
            val[no]=val[esq];
        else
            val[no]=val[dir];
    }
}

int consulta(int no, int i, int j, int A, int B)
{
    if(i>B || j<A)
        return -1;
    if(i>=A and j<=B)
        return val[no];

    int esq = 2*no;
    int dir = 2*no+1;
    int meio = (i+j)/2;

    int resp_esq = consulta(esq, i, meio, A, B);
    int resp_dir = consulta(dir, meio+1, j, A, B);

    if(resp_dir==-1)
        return resp_esq;
    if(resp_esq==-1)
        return resp_dir;

    if(resp_esq>resp_dir)
        return resp_esq;
    else
        return resp_dir;
}

int main()
{
    monta(1, N, 1);
    atualiza(1, 1, N, pos, valor);
    x = consulta(1, 1, N, inicio, fim);

}
```

4.3 Delta-Encoding.cpp

```
// Delta encoding

for(int i=0;i<q;i++)
{
    int l,r,x;
    cin >> l >> r >> x;
    delta[l] += x;
    delta[r+1] -= x;
}

int atual = 0;
```

```cpp

for(int i=0;i<n;i++)
{
    atual += delta[i];
    v[i] += atual;
}
```

4.4 BIT-2D.cpp

```cpp
// BIT 2D

int bit[MAX][MAX];

int sum(int x, int y)
{
    int resp=0;

    for(int i=x;i>0;i-=i&-i)
        for(int j=y;j>0;j-=j&-j)
            resp+=bit[i][j];

    return resp;
}

void update(int x, int y, int delta)
{
    for(int i=x;i<MAX;i+=i&-i)
        for(int j=y;j<MAX;j+=j&-j)
            bit[i][j]+=delta;
}

int query(int x1, y1, x2, y2)
{
    return sum(x2,y2) - sum(x2,y1) - sum(x1,y2) + sum
    (x1,y1);
}
```

4.5 BIT.cpp

```cpp
// (BIT) Fenwick Tree

int bit[MAX];

int soma(int x)
{
    int resp=0;

    // for(int i=x;i>0;i-=i&-i)
    //     resp+=bit[i];

    while(x > 0)
    {
        resp += bit[x];
        x -= (x & -x);
    }

    return resp;
}

int query(int L, R)
{
    return soma(R)-soma(L);
}

void update(int x, int v)
{
    // for(;x<=n;x+=x&-x)
    //       bit[x] += v;

    while(x <= N)
    {
        bit[x] += v;
        x += (x & -x);
    }
}
```

4.6 Union-Find.cpp

```cpp
// Union-Find Functions

int pai[MAX], peso[MAX];

int find(int aux)
{
    if(pai[aux]==aux)
        return aux;
    else
        return pai[aux]=find(pai[aux], pai);
}

void join(int x, int y)
{
    x = find(x);
    y = find(y);

    if(pesos[x]<pesos[y])
        pai[x] = y;
    else if(pesos[x]>pesos[y])
        pai[y] = x;
    else if(pesos[x]==pesos[y])
    {
        pai[x] = y;
        pesos[y]++;
    }
}

int main()
{
    for(int i=1;i<=N;i++)
        pai[i]=i;
}
```

5 STL

5.1 Pair.cpp

```cpp
pair<string, int> P;

cin>>P.first>>P.second;

// Pair of pair

    pair<string, pair<double, double>> P;

    P.first = "Joao";
    P.second.first = 8.2;
    P.second.second = 10;

// Vector of pair

    vector<pair<int, string> > V;
    sort(V.begin(), V.end());

//make.pair()

    P = make_pair("Joao", 10);

    for(int i=1;i<10;i++)
    {
        cin>>a>>b;
        V.push_back(make_pair(a,b));
    }
```

5.2 Set.cpp

```
// Set - Red-Black Trees - O(logn)

set<int> S;

//S.insert()

    S.insert(10); // O(logN)

//S.find()

    if(S.find(3) != S.end())//  O(logN)

//S.erase

    S.erase(10);

    //Outros
    S.clear();
    S.size();
    S.begin();
    S.end();

    p = S.lower_bound(n); // Retorna um ponteiro para o primeiro elemento maior ou igual a n (not less than n)
    p = S.upper_bound(n); // Retorna um ponteiro para o primeiro elemento maior que n (greater than n)


// (set<int>::iterator)

    for(set<int>::iterator it=S.begin(); it!=S.end(); it++)
    {
        cout << *it << " ";
    }
```

5.3 Stack.cpp

```
// Stack

stack<int> pilha;

//pilha.push()

    pilha.push(N);

//pilha.empty()

    if(pilha.empty()==true/false)

//pilha.pop()

    pilha.pop();

//pilha.front()

    p = pilha.top();
```

5.4 Queue.cpp

```
// Queue

queue<int> fila;

//fila.push()

    fila.push(N);

//fila.empty()

    if(fila.empty()==true/false)

//fila.pop()

    fila.pop();

//fila.front()

    p = fila.front();
```

5.5 Priority-Queue.cpp

```
// Priority Queue - O(logn)

priority_queue<int> plista;

//plista.push()

    plista.push(N);

//plista.empty()

    if(plista.empty()==true/false)

//plista.pop()

    plista.pop();

//plista.front()

    p = plista.top();
```

5.6 Map.cpp

```
// Map - Red-Black Trees

map<string, int> M;

//S.insert()

    M.insert(make_pair("Tiago", 18));
    //ou
    M["Tiago"]=18; // O(logN)

//S.find()

    if(M.find("Tiago") != M.end()) // O(logN)

    cout << M["Tiago"] << endl;

//S.erase

    M.erase("Tiago"); // O(logN)


//S.count()

    if(S.count(N))

//Outher

    M.clear();
    M.size();
    M.begin();
    M.end();

// (map<int>::iterator)

    for(map<string,int>::iterator it=M.begin(); it!=M.end(); it++)
    {
        cout << "(" << it->first << ", " << it->
    second << ") ";
    }
```

5.7 Vector.cpp

```
// Vector - Vetor

vector<int> V;
vector<tipo> nome;
vector<tipo> V(n, value);

//push_back()

    V.push_back(2);
    V.push_front(2);

// front() back()

    cout << V.front() << endl;
    cout << V.back() << endl;

//size()

    tamanho = V.size();

//resize()

    V.resize(10);
    V.resize(n, k);

//pop_back()

    V.pop_back();

//clear()

    V.clear();
    sort(V.begin(), V.end());

//upper_bound() e lower_bound()

    vector<int>::iterator low,up;
    low=lower_bound(v.begin(), v.end(), 20);
    up=upper_bound(v.begin(), v.end(), 20);
    cout << "lower_bound at position " << (low- v.
    begin()) << '\n';
    cout << "upper_bound at position " << (up - v.
    begin()) << '\n';

//binary_search()

    if(binary_search(vet.begin(), vet.end(), 15))

//accumulate()

    cout << accumulate(first, last, sum, func) <<
    endl;
    //first - pointer to the first element
    //last - last element
    //sum - inicial value
    //func

    int func(int x, int y)
    {
        //return x*y;
        return x+y;
    }

//partial_sum()

    partial_sum(first, last, vet, func);

    int func(int x, int y)
    {
        //return x*y;
        return x+y;
    }

//assign()
    //Diferente do resize() por mudar o valor de
    todos os elementos do vector

    vector<int> vet;
    vet.assign(N, x);

    vector< vector<int> > vet;
    vet.assign(N, vector<int>());

//sort()

    sort(vet, vet+N, func);

    bool func(Aluno a, Aluno b)
    {
        return a.nota < b.nota; // True caso a venha
    antes de b, False caso contrario
    }
```

6 Math

6.1 Verif-primo.cpp

```
// prime verrification sqrt(N)

long long eh_primo(long long N)
{
    if(N==2)
    {
        return true;
    }
    else if(N==1 or N%2==0)
    {
        return false;
    }
    for(long long i=3;i*i<=N;i+=2)
        if(N%i==0)
            return false;

    return true;
}
```

6.2 Crivo.cpp

```
// Sieve of Eratosthenes

int N;
vector<bool> primos(100010, true);
cin >> N;

primos[0]=false;
primos[1]=false;

for(int i=2;i<=N;i++)
    if(primos[i])
        for(int j=i+i; j<=N; j+=i)
            primos[j]=false;
```

6.3 Kamenetsky.cpp

```
// Number of digits in n! O(1)

#define Pi 3.14159265358979311599796346854
#define Eul 2.71828182845904509079559829842
```

```cpp

long long findDigits(int n)
{
    double x;

    if (n < 0)
        return 0;
    if (n == 1)
        return 1;

    x = ((n * log10(n / euler) + log10(2 * Pi * n)
    /2.0));

    return floor(x) + 1;
}
```

7 Misc

7.1 Bitwise.cpp

```cpp
// Bitwise

    unsigned char a = 5, b = 9; // a = (00000101), b
    = (00001001)

    AND -           a&b   // The result is 00000001
    (1)
    OR -            a|b   // The result is 00001101
    (13)
    XOR -           a^b   // The result is 00001100
    (12)
    NOT -           ~a    // The result is 11111010
    (250)
    Left shift -    b<<1  // The result is 00010010
    (18)
    Right shift -   b>>1  // The result is 00000100
    (4)

    // Exchange two int variables

        a^=b;
        b^=a;
        a^=b;

    // Even or Odd

        (x & 1)? printf("Odd"): printf("Even");

    // Turn on the j-th bit

        int S = 34; //(100010)
        int j = 3;

        S = S | (1<<j);

    // Turn off the j-th bit

        int S = 42; //(101010)
        int j = 1;

        S &= ~(1<<j)

        S == 40 //(101000)

    // Check the j-th element

        int S = 42; //(101010)
        int j = 3;

        T = S & (1<<j); // T = 0

    // Exchange o j-th element

        S ^= (1<<j)

    // Position of the first bit on

        T = (S & (-S))
        T -> 4 bit ligado //(1000)

    // Most significant digit of N


        double K = log10(N);
        K = K - floor(K);
        int X = pow(10, K);

    // Number of digits in N

        X =floor(log10(N)) + 1;

    // Power of two

        bool isPowerOfTwo(int x)
        {
            return x && (!(x&(x-1)));
        }
```

7.2 Complexity.cpp

```
// Complexity

    If n <= 12, the time complexity can be O(n!).
    If n <= 25, the time complexity can be O(2^n).
    If n <= 100, the time complexity can be O(n^4).
    If n <= 500, the time complexity can be O(n^3).
    If n <= 10^4, the time complexity can be O(n^2).
    If n <= 10^6, the time complexity can be O(n log
    n).
    If n <= 10^8, the time complexity can be O(n).
    If n > 10^8, the time complexity can be O(log n)
    or O(1).
```

7.3 Aprox.cpp

```
// Approximation

value-   round() floor() ceil()  trunc()
-----    -----   -----   ----    -----
+2.3     +2.0    +2.0    +3.0    +2.0
+3.8     +4.0    +3.0    +4.0    +3.0
+5.5     +6.0    +5.0    +6.0    +5.0
-2.3     -2.0    -3.0    -2.0    -2.0
-3.8     -4.0    -4.0    -3.0    -3.0
-5.5     -6.0    -6.0    -5.0    -5.0
```

8 Strings