Competitive Programming notes

Druhan Shah (ShockWave)

February 1, 2022

Contents

1 Basic NT **1**
1.1 Prime factorization . . . 1
1.2 Sieve of Eratosthenes . . . 1
1.3 Linear Sieve . . . 1
1.4 Extended Euclidean Algorithm . . . 1

2 Sorting **1**
2.1 Merge Sort (with Inversion count) . . . 1

3 Graphs **2**
3.1 Depth First Traversal (base) . . . 2
3.2 Breadth First Traversal (base) . . . 2
3.3 Breadth First Traversal (for heights on spanning tree) . . . 2

4 Strings **2**
4.1 KMP Algorithm . . . 2

1 Basic NT

1.1 Prime factorization

```
for(int i = 2; i*i<=n; i++)
    if(n%i==0) {
        p.push_back(i);
        while(n%i==0)
            n /= i;
    }
if(n>1) p.push_back(n);
```

1.2 Sieve of Eratosthenes

```
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for(int i = 2; i*i<=n; i++)
    if(is_prime[i]) {
        for(int j = i*i;j<=n; j+=i)
            is_prime[j] = false;
    }
```

1.3 Linear Sieve

```
vector<int> lp(N+1);
vector<int> pr;
FOR(i,2,N+1) {
    if(lp[i]==0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for(int j = 0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j
    ]<=N; j++)
        lp[i*pr[j]] = pr[j];
}
```

1.4 Extended Euclidean Algorithm

```
int gcd(int a, int b, int& x, int& y) {
    if (b==0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a%b, x1, y1);
    x = y1;
    y = x1- y1*(a/b);
    return d;
}
```

2 Sorting

2.1 Merge Sort (with Inversion count)

```
ll mergeSort(int arr[], int array_size);
ll _mergeSort(int arr[], int temp[], int left, int right);
ll merge(int arr[], int temp[], int left, int mid, int right);

ll mergeSort(int arr[], int array_size) {
    int temp[array_size];
    return _mergeSort(arr, temp, 0, array_size - 1);
}
ll _mergeSort(int arr[], int temp[], int left, int right) {
    ll mid, inv_count = 0;
```

```
    if (right > left) {
        mid = (right + left) / 2;
        inv_count += _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid + 1, right);
        inv_count += merge(arr, temp, left, mid + 1, right);
    }
    return inv_count;
}

ll merge(int arr[], int temp[], int left, int mid, int right)
     {
    int i, j, k;
    ll inv_count = 0;

    i = left;
    j = mid;
    k = left;
    while ((i <= mid - 1) && (j <= right)) {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else {
            temp[k++] = arr[j++];
            inv_count = inv_count + (mid - i);
        }
    }
    while (i <= mid - 1)
        temp[k++] = arr[i++];
    while (j <= right)
        temp[k++] = arr[j++];
    FOR(i,left,right+1) arr[i] = temp[i];

    return inv_count;
}
```

3 Graphs

3.1 Depth First Traversal (base)

```
ll dfs(int node, vector<int> adjacency[], bool visited[]) {
    visited[node] = true;
    for(auto i : adjacency[node])
        if(!visited[i]) dfs(i, adjacency, visited);
}
```

3.2 Breadth First Traversal (base)

```
queue<int> tovisit;
ll bfs(bool visited[]) {
    while(!tovisit.empty()) {
        visited[tovisit.front()] = true;
        for(int i : adjacency[tovisit.front()])
            if(!visited[i]) tovisit.push(i);
        tovisit.pop();
    }
}
```

3.3 Breadth First Traversal (for heights on spanning tree)

```
queue<int> tovisit, newvisit;
void bfs(bool visited [], int heights[], int height) {
    while(!tovisit.empty()) {
        heights[tovisit.front()] = min(heights[tovisit.front()
     ], height);
        for (int i : adjacency[tovisit.front()])
            if(!visited[i]) {
                visited[i] = true;
                newvisit.push(i);
            }
        tovisit.pop();
    }
}

void driver() {
    while(!tovisit.empty()) {
        bfs(visited,adjacency,heights,height);
        while(!newvisit.empty()) {
            tovisit.push(newvisit.front());
            newvisit.pop();
        }
    }
    height++;
}
```

4 Strings

4.1 KMP Algorithm

```
void computeLPSArray(string pat, int M, int lps[]);

void KMPSearch(string pat, string txt)
{
    int M = pat.length();
    int N = txt.length();
    int lps[M];
    computeLPSArray(pat, M, lps);

    int i = 0, j = 0;
    while(i<N) {
        if(pat[j]==txt[i]) {
            j++;
            i++;
        }
        if(j==M) {
            cout << i-j << "\n";
            j = lps[j-1];
        }

        // mismatch after j matches
        else if(i<N && pat[j]!=txt[i]) {
            if(j!=0) j = lps[j-1];
            else i++;
        }
    }
}
void computeLPSArray(string pat, int M, int lps[])
{
    int len = 0;
    lps[0] = 0;
    int i = 1;
    while (i<M) {
        if (pat[i]==pat[len]) {
            len++;
            lps[i] = len;
            i++;
        }
        else {
            if (len!=0)
                len = lps[len - 1];
            else {
                lps[i] = 0;
                i++;
            }
        }
    }
}
```