Competitive Programming notes

Druhan Shah (ShockWave)

January 15, 2022

Contents

Macros

```
#define FASTIO() ios_base::sync_with_stdio(false); cin.tie(0)
#define FOR(i,a,b) for(int (i)=(a); i<(b); i++)
#define ROF(i,a,b) for(int (i)=(a); i>(b); i--)
#define MOD 1000000007
#define F first
#define S second
#define all(x) (x).begin(), (x).end()
#define PB push_back
#define MP make_pair
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
```

1 Sorting

1.1 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; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}
```

2 Graphs

2.1 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);
}
```

2.2 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 Strings

3.1 KMP Algorithm

for pattern matching in a string:

```
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++;
            }
        }
    }
}␣␣
```