C++ Cheatsheet

Soumitra Das

November 8, 2021

1 Initial Template

Uncomment line 5-9 if external library is needed.

```cpp
#include <bits/stdc++.h>
using namespace std;
#define FASTio ios::sync_with_stdio(false);cin.tie(
    NULL);
#define DECI fixed<<setprecision(5)
// #include <ext/pb_ds/assoc_container.hpp>
// #include <ext/pb_ds/tree_policy.hpp>
// using namespace __gnu_pbds;
// typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> indexed_set;
// typedef tree<int,null_type,less_equal<int>,
    rb_tree_tag,tree_order_statistics_node_update>
    indexed_multiset;
typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
typedef vector<int> vi;
typedef vector<vector<int>> vvi;
typedef pair<int,int> pii;
typedef priority_queue<int> pqi;
typedef deque<int> di;
#define pb(k) push_back(k)
#define mp(a,b) make_pair(a,b)
#define B begin();
#define E end();
#define nl cout<<"\n"
#define DB(x) {static int testInt=1000;if((testInt--)
    >0)cout<<"(LINE "<<__LINE__<<": VALUE "<<x<<")\t"
    ;}
#define LB {static int testIntx=0;if(testIntx<1000)
    cout<<"(LINE "<<__LINE__<<","<<testIntx+1<<")\t";
    else break;testIntx++;}
#define TA(arr) {int* lLe=(int*)(&arr+1);for(int* xTe=
    arr;xTe!=lLe;xTe++) cout<<*xTe<<" ";nl;}
#define nax 1000000007
/********************************/

int main() {
  FASTio
  int t; cin >> t; while(t--) {
    LB
  }
  return 0;
}
```

2 STL Library

2.1 Containers

vector

deque

list

forward_list

map

unordered_map

multimap

unordered_multimap

set

unordered_set

multiset

unordered_multiset

stack

queue

priority_queue

pair

tuple

tree

2.2 Algorithms

sort

reverse

max_element

min_element

accumulate

count

find

binary_search

lower_bound

upper_bound

next_permutation

prev_permutation

partition

stable_partition

rotate

min

max

swap

__gcd

__builtin_popcount

3 Algorithms

3.1 Fibonacci numbers

if F_n is the n'th Fibonacci number, where $F_0 = 0$ and $F_1 = 1$, then

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$$

for any $n, k \in \mathbb{N}$.

3.2 Geometric Transformation of points

Point (x, y, z) can be transformed by matrix multiplication

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \times \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} x' & y' & z' & 1 \end{bmatrix}$$

Where (x', y', z') is our answer. If we call the 4×4 matrix as X, then for shifting x by a co-ordinate, y by b and z by c co-ordinate,

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}$$

Instead of shifting, for scaling

$$X = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And finally, for rotating θ degrees around the x axis following the right-hand rule (counter-clockwise direction)

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For 2D rotation of (x, y) by θ degree counterclockwise,

$$\begin{bmatrix} x & y \end{bmatrix} \times \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}$$

Where (x', y') is our answer.

3.3 Extended Euclidean Algorithm

Returns the gcd of a and b with $ax + by = \gcd(a, b)$.

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

3.4 Binary Search

Returns the index of x in array a.

```
int bin_search(int a[],int n,int x) {
    int l=0,r=n-1;
    while(l<=r){
        int k=(l+r)/2;
        if(a[k]==x){
            return k;
        }
        if(a[k]>x) r=k-1;
        else l=k+1;
    }
    return -1;
}
```

3.5 Processing All Subset

Processes subset of array a. Initially $k = 0$ and s empty.

```
void all_subset(int a[],int n,int k,vector<int> s) {
    if(k==n){
        // process subset
    }
    else{
        all_subset(a,n,k+1,s);
        s.push_back(a[k]);
        all_subset(a,n,k+1,s);
        s.pop_back();
    }
}
```

3.6 Processing All Permutation

Processes all permutation of array a, all element should be distinct. bm is an boolean array with length n, initially all element is $false$.

```
void all_permutation(int a[],int n,vector<int> p,int
    bm[]) {
    if(p.size()==n) {
        // process permutation
    }
    else {
        for(int i=0;i<n;i++) {
            if(bm[i]) continue;
            bm[i] = true;
            p.push_back(a[i]);
            all_permutation(a,n,p,bm);
            bm[i] = false;
            p.pop_back();
        }
    }
}
```

3.7 Miller Rabin Primality Test

Checks whether n prime or not, n must be in int range. Time complexity $O(\log n)$

```
typedef long long ll;

int binpower(int base,int e,int mod) {
    int result=1;
    base%=mod;
    while(e){
        if(e&1)
            result=(ll)result*base%mod;
        base=(ll)base*base%mod;
        e>>=1;
    }
    return result;
}
bool check_composite(int n,int a,int d,int s) {
    int x=binpower(a,d,n);
    if(x==1 || x==n-1) return false;
    for(int r=1;r<s;r++){
        x=(ll)x*x%n;
        if(x ==n-1) return false;
    }
    return true;
```

```cpp
22 };
23 bool MillerRabin(int n) {
24     if (n<2) return false;
25     int r=0;
26     int d=n-1;
27     while((d&1)==0){
28         d>>=1;
29         r++;
30     }
31     for(int a:{2,3,5,7}){
32         if(n==a) return true;
33         if(check_composite(n,a,d,r)) return false;
34     }
35     return true;
36 }
```

3.8   DFS algorithm

Runs DFS on a graph with adjacency matrix e, initially all elements of bm false.

```cpp
1 void dfs(vector<vector<int>> &e, vector<bool> &bm, int
    v) {
2     bm[v] = true;
3     //process vertex v
4     for (int i: e[v]) {
5         if (!bm[i]) DFS(e,bm,i);
6     }
7 }
```

3.9   BFS algorithm

Runs BFS on a graph with adjacency matrix e, starting point s, vertex number n.

```cpp
1 void bfs(vector<vector<int>> e,int n, int s)
2 {
3     vector<bool> bm(n);
4     for(int i=0;i<n;i++) bm[i]=false;
5     queue<int> q;
6     visited[s] = true;
7     q.push(s);
8     while(!q.empty())
9     {
10         s=q.top();
11         //process vertex s
12         q.pop();
13         for (int i: e[s])
14         {
15             if (!bm[i]) {
16                 bm[i] = true;
17                 q.push(i);
18             }
19         }
20     }
21 }
```

3.10   modular inverse

Finds modular multiplicative inverse from 1 to n inclusive mod m.

```cpp
1 vector<int> mod_inv(int n, int m) {
2     vector<int> inv(n+1);
3     inv[0]=-1;inv[1]=1;
4     for(int i=2; i<=n; i++) inv[i]=m-((m/i)*inv[m%i])%
    m;
5     return inv;
6 }
```

4   Useful Results

4.1   matrix

The element val of this struct contains the value of the elements of the matrix, where $val[i][j]$ represents the value in i'th row and j'th column.

```cpp
1 struct matrix {
2   vector<vector<int>> val;
3   matrix(int n) {
4     vector<int> temp(n,0);
5     for(int i=0;i<n;i++) val.push_back(temp);
6   }
7   matrix operator+(matrix x) {
8     matrix t_matrix(val.size());
9     for(int i=0;i<val.size();i++) for(int j=0;j<val.
    size();j++) {
10       t_matrix.val[i][j]=val[i][j]+x.val[i][j];
11     }
12     return t_matrix;
13   }
14   matrix operator-(matrix x) {
15     matrix t_matrix(val.size());
16     for(int i=0;i<val.size();i++) for(int j=0;j<val.
    size();j++) {
17       t_matrix.val[i][j]=val[i][j]-x.val[i][j];
18     }
19     return t_matrix;
20   }
21   matrix operator*(matrix x) {
22     matrix t_matrix(val.size());
23     for(int i=0;i<val.size();i++) for(int j=0;j<val.
    size();j++) {
24       int temp=0;
25       for(int k=0;k<val.size();k++) temp+=val[i][k]*(x
    .val[k][j]);
26       t_matrix.val[i][j]=temp;
27     }
28     return t_matrix;
29   }
30 };
```

4.2   Finding directed path with fixed length

Create the adjacency matrix and raise it's power to k, cell (u,v) will give the number of distinct path with length k connecting vertex u and v (direction from u to v).

4.3   Gray Code

Gray code is a binary numeral system where two successive values differ in only one bit.
For example, the sequence of Gray codes for 3-bit numbers is: $000, 001, 011, 010, 110, 111, 101, 100$, so $G(4) = 6$. Function for finding n'th gray code:

```cpp
1 int g (int n) {
2     return n^(n>>1);
3 }
```