# C++ Cheatsheet

Soumitra Das

November 27, 2021

## 1 Initial Template

Uncomment line 5-9 if external library is needed.

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  #define FASTio ios::sync_with_stdio(false);cin.tie(NULL);
4  #define DECI fixed<<setprecision(5)
5  
6  // #include <ext/pb_ds/assoc_container.hpp>
7  // #include <ext/pb_ds/tree_policy.hpp>
8  // using namespace __gnu_pbds;
9  // typedef tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update> indexed_set;
10 // typedef tree<int,null_type,less_equal<int>,rb_tree_tag,tree_order_statistics_node_update> indexed_multiset;
11 
12 typedef long long ll;typedef unsigned long long ull;typedef long double ld;
13 typedef vector<int> vi;typedef vector<vector<int>> vvi;typedef deque<int> di;
14 typedef map<int,int> mii;
15 typedef pair<int,int> pii;typedef tuple<int,int,int> tiii;
16 typedef priority_queue<int> pqi;typedef priority_queue<int,vector<int>,greater<int>> pqgi;
17 typedef set<int> si;typedef multiset<int> msi;
18 #define pb(k) push_back(k)
19 #define mp(a,b) make_pair(a,b)
20 #define B begin()
21 #define E end()
22 #define F first
23 #define S second
24 #define nl cout<<"\n"
25 /***************Debugging tools are below*****************/
26 #define LC cout<<"line("<<__LINE__<<") ";
27 #define DB(x) {static int _ti_=1000;if((_ti_--)>0)cout<<#x<<": "<<x<<"\n";}
28 #define LB {static int _tx_=0;if(_tx_>=1000) {cout<<"inf loop\n" ;break;}_tx_++;}
29 #define TA(a) {int* n=(int*)(&a+1);cout<<#a<<": ";for(int* i=a;i!=n;i++) cout<<*i<<" ";nl;}
30 #define nax 1000000007
31 /*******************************************************/
32 int main() {
33   FASTio
34   int t; cin >> t; while(t--) {
35     LB
36     LC DB(t)
37   }
38   return 0;
39 }
```

## 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 $4 \times 4$ 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 $\theta$ 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 $\theta$ 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;
};
bool MillerRabin(int n) {
    if (n<2) return false;
    int r=0;
    int d=n-1;
    while((d&1)==0){
        d>>=1;
        r++;
    }
    for(int a:{2,3,5,7}){
        if(n==a) return true;
        if(check_composite(n,a,d,r)) return false;
    }
    return true;
}
```

## 3.8 DFS algorithm

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

```
void dfs(vector<vector<int>> &e, vector<bool> &bm, int v) {
    bm[v] = true;
    //process vertex v
    for (int i: e[v]) {
        if (!bm[i]) DFS(e,bm,i);
    }
}
```

## 3.9 BFS algorithm

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

```
void bfs(vector<vector<int>> e,int n, int s)
{
    vector<bool> bm(n);
    for(int i=0;i<n;i++) bm[i]=false;
    queue<int> q;
    visited[s] = true;
    q.push(s);
    while(!q.empty())
    {
        s=q.top();
        //process vertex s
        q.pop();
        for (int i: e[s])
        {
            if (!bm[i]) {
                bm[i] = true;
                q.push(i);
            }
        }
    }
}
```

## 3.10 modular inverse

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

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

## 3.11 Fast Fourier Transformation

For finding polynomial values at roots of unity for polynomial with co-efficient $a$, set $invert = false$, for finding co-efficient form roots of unity, set $invert = true$. Size of $a$ has to be $2^k$ for some $k \in \mathbb{N}$. See here for more details.

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> &a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;

    vector<cd> a0(n/2), a1(n/ 2);
    for (int i = 0; 2*i <n; i++) {
        a0[i] = a[2*i];
        a1[i] = a[2*i+1];
    }
    fft(a0, invert);
    fft(a1, invert);

    double ang = 2*PI/n*(invert ? -1 : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2*i < n; i++) {
        a[i] = a0[i] + w*a1[i];
        a[i + n/2] = a0[i] - w*a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n/2] /= 2;
        }
        w *= wn;
    }
}
```

# 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.

```
struct matrix {
  vector<vector<int>> val;
  matrix(int n) {
    vector<int> temp(n,0);
    for(int i=0;i<n;i++) val.push_back(temp);
  }
  matrix operator+(matrix x) {
    matrix t_matrix(val.size());
    for(int i=0;i<val.size();i++) for(int j=0;j<val.size();j++) {
      t_matrix.val[i][j]=val[i][j]+x.val[i][j];
    }
    return t_matrix;
  }
  matrix operator-(matrix x) {
    matrix t_matrix(val.size());
    for(int i=0;i<val.size();i++) for(int j=0;j<val.size();j++) {
      t_matrix.val[i][j]=val[i][j]-x.val[i][j];
    }
    return t_matrix;
  }
  matrix operator*(matrix x) {
    matrix t_matrix(val.size());
    for(int i=0;i<val.size();i++) for(int j=0;j<val.size();j++) {
      int temp=0;
      for(int k=0;k<val.size();k++) temp+=val[i][k]*(x.val[k][j]);
```

```
        t_matrix.val[i][j]=temp;
    }
    return t_matrix;
  }
};
```

## 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:

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