Contents

1 Basic

1.1 bat

==== w.bat ===

```
notepad++ %1
```

==== i.bat ===

```
w %p%_%1.txt
```

==== s.bat ===

```
set p=%1
```

==== r.bat ===

```
cls
g++ %p%.cpp -Wall -Wextra -Wshadow -std=c++11 -o %p%
if /i "%ERRORLEVEL%" == "0" @for %%i in (%p%_*.txt) do
    @echo ====== %%i ====== & %p% < %%i
```

1.2 vimrc

=== .vimrc ===

```
set et nu cin ls=2 ts=4 sw=4 sts=4
syntax on

nn <F7> :w <bar> :!~/.script addin %< <CR>
nn <F8> :w <bar> :!~/.script %:e %< <CR>
```

=== .script ===

```
#!/bin/bash
CF="-std=c++11 -fsanitize=undefined -D FOX"
WF="-Wall -Wextra -Wshadow -pedantic"
PN=$(echo $2 | sed 's/\..*$//')
cpp(){
    g++ $1.cpp $CF $WF -o $1 && run ./$1
}
py(){
    run "python $1.py"
}
addin(){
    read -p "case name: " CASE && vim $PN\_$CASE.in
}
run(){
    for i in "$PN"_*.in
    do
        echo "======== $i ========"
        bash -c "$1" < $i
    done
}
echo "=========================v"
"$1" "$2"
echo "=========================^"
```

2 Graph

2.1 Matching

2.1.1 Hungarian

```
// Maximum Cardinality Bipartite Matching
// Worst case O(nm)

struct Graph{
    static const int MAXN = 5003;
    vector<int> G[MAXN];
    int n, match[MAXN], vis[MAXN];

    void init(int _n){
        n = _n;
        for (int i=0; i<n; i++) G[i].clear();
    }
```

```
    bool dfs(int u){
        for (int v:G[u]){
            if (vis[v]) continue;
            vis[v]=true;
            if (match[v]==-1 || dfs(match[v])){
                match[v] = u;
                match[u] = v;
                return true;
            }
        }
        return false;
    }

    int solve(){
        int res = 0;
        memset(match,-1,sizeof(match));
        for (int i=0; i<n; i++){
            if (match[i]==-1){
                memset(vis,0,sizeof(vis));
                if ( dfs(i) ) res++;
            }
        }
        return res;
    }
} graph;
```

3 Math

3.1 ax+by = gcd

```
pair<int,int> extgcd(int a, int b){
    if (b==0) return {1,0};
    int k = a/b;
    pair<int,int> p = extgcd(b,a-k*b);
    return { p.second, p.first - k*p.second };
}
```

3.2 FFT

```
// use llround() to avoid EPS
typedef double Double;
const Double PI = acos(-1);

// STL complex may TLE
typedef complex<Double> Complex;
#define x real()
#define y imag()

template<typename Iter> // Complex*
void BitReverse(Iter a, int n){
    for (int i=1, j=0; i<n; i++){
        for (int k = n>>1; k>(j^=k); k>>=1);
        if (i<j) swap(a[i],a[j]);
    }
}

template<typename Iter> // Complex*
void FFT(Iter a, int n, int rev=1){ // rev = 1 or -1
    assert( (n&(-n)) == n ); // n is power of 2
    BitReverse(a,n);
    Iter A = a;

    for (int s=1; (1<<s)<=n; s++){
        int m = (1<<s);

        Complex wm( cos(2*PI*rev/m), sin(2*PI*rev/m) );
        for (int k=0; k<n; k+=m){
            Complex w(1,0);
            for (int j=0; j<(m>>1); j++){
                Complex t = w * A[k+j+(m>>1)];
                Complex u = A[k+j];
                A[k+j] = u+t;
                A[k+j+(m>>1)] = u-t;
                w = w*wm;
            }
        }
    }

    if (rev==-1){
        for (int i=0; i<n; i++){
            A[i] /= n;
        }
    }
}
```

3.3 inverse

```
const int MAXN = 1000006;
int inv[MAXN];
void invTable(int bound, int p){
    inv[1] = 1;
    for (int i=2; i<bound; i++){
        inv[i] = (long long)inv[p%i] * (p-p/i) %p;
    }
}

int inv(int b, int p){
    if (b==1) return 1;
    return (long long)inv(p%b,p) * (p-p/b) %p;
}
```

3.4 Karatsuba

```
// N is power of 2
template<typename Iter>
void DC(int N, Iter tmp, Iter A, Iter B, Iter res){
    fill(res,res+2*N,0);
    if (N<=32){
        for (int i=0; i<N; i++){
            for (int j=0; j<N; j++){
                res[i+j] += A[i]*B[j];
            }
        }
        return;
    }
    int n = N/2;
    auto a = A+n, b = A;
    auto c = B+n, d = B;
    DC(n,tmp+N,a,c,res+2*N);
    for (int i=0; i<N; i++){
        res[i+N] += res[2*N+i];
        res[i+n] -= res[2*N+i];
    }
    DC(n,tmp+N,b,d,res+2*N);
    for (int i=0; i<N; i++){
        res[i] += res[2*N+i];
        res[i+n] -= res[2*N+i];
    }

    auto x = tmp;
    auto y = tmp+n;
    for (int i=0; i<n; i++) x[i] = a[i]+b[i];
    for (int i=0; i<n; i++) y[i] = c[i]+d[i];
    DC(n,tmp+N,x,y,res+2*N);
    for (int i=0; i<N; i++){
        res[i+n] += res[2*N+i];
    }
}
// DC(1<<16,tmp.begin(),A.begin(),B.begin(),res.begin()
    );
```