# Team Notebook

anarap+1 - UTN FRSF

April 10, 2024

## Contents

# 1 C++ utils

```cpp
// 1- (mt19937_64 for 64-bits version)
mt19937 rng(
    chrono::steady_clock::now().time_since_epoch().count());
shuffle(v.begin(), v.end(), rng); // vector random shuffle
// 2- Pragma
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
// 3- Custom comparator for set/map
struct comp {
  bool operator()(const double& a, const double& b) const {
    return a + EPS < b;
  }
};
set<double, comp> w; // or map<double,int,comp>
// 4- Iterate over non empty subsets of bitmask
for (int s = m; s; s = (s - 1) & m) // Decreasing order
for (int s = 0; s = s - m & m;)  // Increasing order
// 5- Other bits operations
int __builtin_popcount(unsigned int x) // # of bits on in x
int __builtin_popcountll(unsigned long long x) // ll version
int __builtin_ctz(unsigned int x) //# of trailing 0 (x != 0)
int __builtin_clz(unsigned int x) // # of leading 0 (x != 0)
v = (x & (-x)) // Find the least significant bit that is on
// 6- Input
inline void Scanf(int& a) { // only positive ints
  char c = 0;
  while (c < 33) c = getc(stdin);
  a = 0;
  while (c > 33) a = a * 10 + c - '0', c = getc(stdin);
}
```

# 2 DQ dp

```cpp
vector<vector<int>> dp; //maybe replace dim of K with only 2
int n;
// d&q DP: go down the range [l,r) like merge sort, but also
// making sure to iterate over [from,to) in each step, and
// spliting the [from,to) in 2 parts when goind down:
// [from, best] and [best, to)
void solve(int l, int r, int k, int from, int to) {
 if(l >= r) return;
 int cur = (l+r)/2;
 int bestpos = cur-1;
 int best = INF; // assumes we want to minimize cost
 forr(i,from,min(cur, to)) {
  // cost function that usually depends on dp[i][k-1]
  int c = fcost(i, k);
  if(c < best) best = c, bestpos = i;
 }
 dp[cur][k] = best;
 solve(l, cur, k, from, bestpos+1);
 solve(cur+1, r, k, bestpos, to);
}
```

# 3 Mo's

```cpp
// Commented code should be used if updates are needed
int n, sq, nq; // array size, sqrt(array size), #queries
struct Qu {    //[l, r)
  int l, r, id;
  // int upds; // # of updates before this query
};
Qu qs[MAXN];
ll ans[MAXN]; // ans[i] = answer to ith query
// struct Upd{
//   int p, v, prev; // pos, new_val, prev_val
// };
// Upd vupd[MAXN];

// Without updates
bool qcomp(const Qu& a, const Qu& b) {
  if (a.l / sq != b.l / sq) return a.l < b.l;
  return (a.l / sq) & 1 ? a.r < b.r : a.r > b.r;
}
// With updates
// bool qcomp(const Qu &a, const Qu &b){
//   if(a.l/sq != b.l/sq) return a.l<b.l;
//   if(a.r/sq != b.r/sq) return a.r<b.r;
//   return a.upds < b.upds;
// }

// Without updates: O(n^2/sq + q*sq)
// with sq = sqrt(n): O(n*sqrt(n) + q*sqrt(n))
// with sq = n/sqrt(q): O(n*sqrt(q))
//
// With updates: O(sq*q + q*n^2/sq^2)
// with sq = n^(2/3): O(q*n^(2/3))
// with sq = (2*n^2)^(1/3) may improve a bit
void mos() {
  forn(i, nq) qs[i].id = i;
  sq = sqrt(n) + .5; // without updates
  // sq=pow(n, 2/3.0)+.5; // with updates
  sort(qs, qs + nq, qcomp);
  int l = 0, r = 0;
  init();
  forn(i, nq) {
    Qu q = qs[i];
    while (l > q.l) add(--l);
    while (r < q.r) add(r++);
    while (l < q.l) remove(l++);
    while (r > q.r) remove(--r);
    // while(upds<q.upds){
    //   if(vupd[upds].p >= l && vupd[upds].p < r)
    //     remove(vupd[upds].p);
    //   v[vupd[upds].p] = vupd[upds].v; // do update
    //   if(vupd[upds].p >= l && vupd[upds].p < r)
    //     add(vupd[upds].p);
    //   upds++;
    // }
    // while(upds>q.upds){
    //   upds--;
    //   if(vupd[upds].p >= l && vupd[upds].p < r)
    //     remove(vupd[upds].p);
    //   v[vupd[upds].p] = vupd[upds].prev; // undo update
    //   if(vupd[upds].p >= l && vupd[upds].p < r)
    //     add(vupd[upds].p);
    // }
    ans[q.id] = get_ans();
  }
}
```

# 4 Python example

```python
import sys, math
input = sys.stdin.readline
############ ---- Input Functions ---- ############
def inp():
    return(int(input()))
def inlt():
    return(list(map(int,input().split())))
def insr():
    s = input()
    return(list(s[:len(s) - 1]))
def invr():
    return(map(float,input().split()))

n, k = inlt() # read two numbers in a line
```

# 5 Template

```cpp
#include <bits/stdc++.h>
#define forr(i, a, b) for (int i = (a); i < (b); i++)
#define forn(i, n) forr(i, 0, n)
#define dforn(i, n) for (int i = (n) - 1; i >= 0; i--)
#define forall(it,v) for(auto it=v.begin();it!=v.end();it++)
#define sz(c) ((int)c.size())
#define rsz resize
#define pb push_back
#define mp make_pair
#define lb lower_bound
#define ub upper_bound
#define fst first
#define snd second
using namespace std;
typedef long long ll;
typedef pair<int, int> ii;

int main() {
#ifdef ANARAP
  freopen("input.in", "r", stdin);
#endif
  ios::sync_with_stdio(false);
  cin.tie(NULL); cout.tie(NULL);
  return 0;
}
```

# 6 Test generator

```python
# python3 test_generator.py A B (A B are compiled files)
# A and B must READ FROM STANDARD INPUT, not from file.
import sys, subprocess
from datetime import datetime
from random import randint, seed

def buildTestCase(): # example of trivial "a+b" problem
 a = randint(1,100)
 b = randint(1,100)
 return f"{a} {b}\n"

seed(datetime.now().timestamp())
ntests = 100 # change as wanted
sol1 = sys.argv[1]
sol2 = sys.argv[2]
# Sometimes it's a good idea to use extra arguments to then
# be passed to 'buildTestCase' for "shaping" your tests
for curtest in range(ntests):
 test_case = buildTestCase()
 # Here the test is executed and outputs are compared
 print("running... ", end='')
 ans1 = subprocess.check_output(f"./{sol1}",
  input=test_case.encode('utf-8')).decode('utf-8')
 ans2 = subprocess.check_output(f"./{sol2}",
  input=test_case.encode('utf-8')).decode('utf-8')
 if ans1 == ans2:
  assert ans1 != "", 'ERROR? ans1 = ans2 = empty ("")'
  print("OK")
 else:
  print("FAILED!")
  print(test_case)
  print(f"ans from {sol1}:\n{ans1}")
  print(f"ans from {sol2}:\n{ans2}")
  break
```