Clean Doc 15/03/2018

Table des matières

1. Implementation

```
assert
binary search
bit
iterator operations
map
priorityQ
set unordered_set
slice
slice (gslice)
sort
sort (index sort)
stream
switch case
tuple
var
vector
```

```
//////////////////////////////
# assert
//////////////////////////////
assert(length >= 0); // die if length
is negative.
assert(length >= 0 && "Whoops, length
can't possibly be negative! (didn't we
just check 10
lines ago?) Tell jsmith");
// BAD
assert(x++);
// GOOD
assert(x);
x++;
// Watch out! Depends on the function:
assert(foo());
// Here's a safer way:
int ret = foo();
assert(ret);
////////////////////////////
int &ret=mem[i][w0];
if( ret !=-1 )return ret;
return ret=( ... ;
auto lambda = [](int x, int y) {return
x + y;}; // C++11 --- had to specify
type of x and y
```

```
//////////////////////////////////////
//////////////////////////////
Binary
//////////////////////////////////////
//////////////////////////////
This is called reducing the original
problem to a decision (yes/no)
problem.
call the main theorem states that
binary search can be used if and only
if for all x in S, p(x) implies p(y)
for all y > x.
getting a yes answer for some
potential solution x means that you'd
also get a yes answer for any element
after x. Similarly, if you got a no
answer, you'd get a no answer for any
element before x


bool p(int j){
    return (get<0>(g[i][j])) > wp ;
}


int bs( int lo, int hi, bool (*p)(int)
){
while (lo < hi)
    int mid = lo + (hi-lo)/2;    //
peut causer des prob avec un tab 2
elmt {no, yes} à verif :/
    if (p(mid))
        hi = mid;
    else
        lo = mid+1;

    if (!p(lo))
        return -1;//complain:  p(x) is
false for all x in S!
!!!!!!!!!!!!!!!!!

    return lo; // lo is the least x
for which p(x) is true
}
----
Implementing binary search on reals is
usually easier than on integers,
because you don't
need to watch out for how to move
bounds:
binary_search(lo, hi, p){
while we choose not to terminate:
mid = lo + (hi-lo)/2
```

```
if p(mid) == true:
hi = mid
else:
lo = mid
return lo // lo is close to the border
between no and yes
}
just use a few hundred iterations,
this will give you the best possible
precision without
too much thinking. 100 iterations will
reduce the search space to
approximately (hi-lo)/2^
100
Binary search in standard libraries
C++'s Standard Template Library
implements binary search in algorithms
lower_bound,
upper_bound, binary_search and
equal_range, depending exactly on what
you need to do.


#######################################
###################
# BIT
#######################################
###################
//print
cout<<  bitset<20>(x)  <<endl;

// IMG 1 @01
//Exemple
unsigned int x = 4376;
00000000000000000001000100011000
```

__builtin_clz(x)
count leading 0-bit

x = 00000000000000000001000100011000

__builtin_ctz(x)
count trailing 0-bits

__builtin_ffs(x)
find first set
1er 1-bit à 1
le moin signifiant
1-based

__builtin_popcount(x)
nombre total de 1-bits

```
__builtin_clz(x) = 19   count leading
0-bits            x==0 => returns an
undefined value.
__builtin_clzll(x)=51   cas x ull (on
ajoute le suffixe ll pour travailler
avec les ull)
__builtin_ctz(x) = 3    count trailing
0-bits
__builtin_ffs(x) = 4    find first set
(1er bit à 1)
__builtin_popcount(x)=4 nombre de bit à
1


//Code Gray
printf("%d\n", k^(k>>1));//Original To
code Gray

// code Gray  To Original
long grayInverse(long n) {
      long ish=1, ans=n, idiv;
      while(1) {
            idiv = ans >> ish;
            ans ^= idiv;
            if (idiv <= 1 || ish == 32)
return ans;
            ish <<= 1; // double le nb
de shifts la prochaine fois
      }
}

#######################################
#Iterator operations:
#######################################
advance
Advance iterator (function template )
distance
Return distance between iterators
(function template )
begin
Iterator to beginning (function
template )
end
Iterator to end (function template )
prev
Get iterator to previous element
(function template )
next
Get iterator to next element (function
template )

#######################################
###################
map
#######################################
########################
auto s=m.find(make_pair(k, t1));
//map.find("key")
if ( s !=m.end() )// found
{//cerr<<"("<< get<0>(s->first) <<" "
<< get<1>(s->first) <<" "<<
(*s).second <<")";
return s->second;//valeur ~
}

#######################################
#PriorityQ
#######################################
//{
priority_queue<int ,
vector<pair<int,int>>,
greater<pair<int,int>> >
trié selon, tiré dans l ordre DESC (
default )
v
priority_queue<int , vector<
pair<int,int> >, greater<int> > q ;
q.push(make_pair(1,2));
q.push(make_pair(10,4));
q.push(make_pair(0,3));
pair<int,int> s;
s= q.top() ;
cout << s.fi;// val de priority
q.pop();
```

```
s= q.top() ;
cout << s.fi;
q.pop();
s= q.top() ;
cout << s.fi;
q.pop();
trié selon 1ere var , tiré dans
l'ordre CROISSANT
v
priority_queue<int ,
vector<pair<int,int>>,
greater<pair<int,int>> > q ;
//}



#######################################
############################
Set
#######################################
############################
set<int> s( vec.begin(), vec.end() );
vec.assign( s.begin(), s.end() );

multiset.erase(it) mara bark
multiset.erase(40) efface tt les occ
de 40
insertion set : O(log n) or const with
a hint
insertion unsorted_set O(n)
unsorted multiset O(1) -> O(n) worst
------
unordered_set<int> s(53);// n'affecte
pas le nb 53 à la 1ere case nooooooo!
unordered_set( size_type bucket_count,
...)
// erasing from set
int main ()
{
std::set<int> myset;
std::set<int>::iterator it;
// insert some values:
for (int i=1; i<10; i++)
myset.insert(i*10); // 10 20 30 40 50
60 70 80 90
it = myset.begin();
++it; // "it" points now to 20
myset.erase (it);
myset.erase (40);
it = myset.find (60);
myset.erase (it, myset.end());
std::cout << "myset contains:";
for (it=myset.begin();
it!=myset.end(); ++it)
std::cout << ' ' << *it;
std::cout << '\n';
return 0;
}
// set::lower_bound/upper_bound
int main ()
{
std::set<int> myset;
std::set<int>::iterator itlow,itup;
for (int i=1; i<10; i++)
myset.insert(i*10); // 10 20 30 40 50
60 70 80 90
itlow=myset.lower_bound (30); // ^
itup=myset.upper_bound (60); // ^
myset.erase(itlow,itup); // 10 20 70
80 90
std::cout << "myset contains:";
for (std::set<int>::iterator
it=myset.begin(); it!=myset.end();
++it)
std::cout << ' ' << *it;
std::cout << '\n';
return 0;
}



#######################################
####################
#slice
#######################################
####################
int main ()
{
  std::valarray<int> foo (12);
  for (int i=0; i<12; ++i)
foo[i]=i*100;
int idx_start = 2;
int size = 3;
int pas = 4;
  std::valarray<int> bar =
foo[std::slice(idx_start,size,pas)];

  std::cout << "slice(2,3,4):";
  for (std::size_t n=0; n<bar.size();
n++)
        std::cout << ' ' << bar[n];
  std::cout << '\n';

  return 0;
}
#result ::     slice(2,3,4): 200 600
1000

#######################################
####################
#gslice    ;*
#######################################
####################



#######################################
####################
#sort (index sort)
#######################################
####################
int a[100], p[100];// receive input
for (int i = 0; i < n; ++i)
scanf("%d", &a[i]), p[i] = i;
sort(p, p+n, [=](int i, int j) {
return a[i] < a[j]; });
#######################################
####################
#sort:
#######################################
####################
// using default comparison (operator
<):
std::sort (myvector.begin(),
myvector.begin()+4);
// using function as comp
std::sort (myvector.begin()+4,
myvector.end(), myfunction);
//{
vector<double> tableau;
tableau.push_back(8);// comme
_Array_Add()
tableau.pop_back(); //Et hop ! la
dernière case a sauté
tableau.size() // Ubound(tab)
//Une fonction recevant un tableau
d'entiers en argument
void fonction(vector<int> a)
void fonction(vector<int> const& a)
vector<double> encoreUneFonction(int
a)
```

```
//-----------------------------------
-----
//Notez qu'il est aussi possible de
créer des tableaux multi-dimensionnels
de taille
variable en utilisant les vectors.
Pour une grille 2D d'entiers, on devra
écrire :
vector<vector<int> > grille;
grille.push_back(vector<int>(5)); //On
ajoute une ligne de 5 cases à notre
grille
grille.push_back(vector<int>(3,4));
//On ajoute une ligne de 3 cases
contenant chacune le
nombre 4 à notre grille
//Chaque ligne peut donc avoir une
longueur différente. On peut accéder à
une ligne en utilisant les crochets:
grille[0].push_back(8); //Ajoute une
case contenant 8 à la première ligne
du tableau
grille[2][3] = 9; //Change la valeur
de la cellule (2,3) de la grille
/*
Les tableaux multi-dimensionnels
utilisant des vector ne sont pas la
meilleure manière
d'accéder efficacement à la mémoire et
ne sont pas très optimisés. On
préférera donc
utiliser des tableaux multi-
dimensionnels statiques à moins que le
fait de pouvoir changer
la taille de la grille en cours de
route soit un élément essentiel.
*/
vect.clear(); // reinitialise tab
(size 0)
//A reallocation is not guaranteed to
happen, and the vector capacity is not
guaranteed to change due to calling
this function. A typical alternative
that forces a reallocation is to
vector<T>().swap(x); // clear x
reallocating
//}
```

```
#####################################
###################
# STREAM
#####################################
###################
//prend vect resultat qu'elle va
reinitialiser
// return vect.size
char readline(vector<ll> &r)
{
std::string line;
std::getline(std::cin, line); //  <---
-------------------------
std::istringstream line_buffer(line);
//std::vector<ll> r; //resultat à
retourner
ll x;
r.clear(); // vide tab
op = '\0';
if (line_buffer.peek()!=EOF)
{line_buffer >> op;
//r.push_back( (ll) x);
}
while(line_buffer.peek()!=EOF )
        {
            line_buffer >> x;
            line_buffer >> std::ws;
// eat up any leading white spaces



            //cout << x<<" , ";//
traitement
            r.push_back(x);
        }

        return op;
}



#Switch case
#####################################
#
//galere ce truc
int main()
{ int i = 2;
switch (i) {
case 1: std::cout << "1";
case 2: std::cout << "2"; //execution
starts at this case label
case 3: std::cout << "3";
case 4:
case 5: std::cout << "45";
break; //execution of subsequent
statements is terminated
case 6: std::cout << "6";
}
std::cout << '\n';
switch (i) {
case 4: std::cout << "a";
default: std::cout << "d"; //there are
no applicable constant_expressions
//therefore default is executed
}
std::cout << '\n';
switch (i) {case 4: std::cout << "a";
//nothing is executed
}
// when enumerations are used in a
switch statement, many compilers
// issue warnings if one of the
enumerators is not handled
enum color {RED, GREEN, BLUE};
switch(RED) {
case RED: std::cout << "red\n"; break;
case GREEN: std::cout << "green\n";
break;
case BLUE: std::cout << "blue\n";
break;
-3-
C:\Users\Karim\Desktop\base.cpp lundi
13 février 2017 22:39
}
// pathological examples
// the statement doesn't have to be a
compound statement
switch(0)
std::cout << "this does nothing\n";
// labels don't require a compound
statement either
switch(int n = 1)
case 0:
case 1: std::cout << n << '\n';
// Duff's Device:
http://en.wikipedia.org/wiki/Duff's_de
vice
}//Exemple 2
char keystroke = getch();
```

```
switch( keystroke ) {
case 'a':
case 'b':
KeyABPressed();
break;
default:
UnknownKeyPressed();
break;
}


#tuple
#####################################
#
tie(a, std::ignore, c) = oTuple;
// ne sont pas syncho: changer a ou
oTuple n'affectera pas l'autre


#####################################
###
#VAR
register int x; // le compilateur
choisira surement de placer la var
dans le registre
memset (str,'-',6);
void func ( void (*f)(int) );




#####################################
####################
#VECTOR
#####################################
####################
segmentation fault core dumped -->
acces en dehors du vector
vector<int> some_list { 1, 2, 3, 4, 5
}; // oui c++11
//vector< vector< bool > >
vector< vector< bool > > myvector(
cols, vector<bool>( rows, false ) );
(!) .size est un size_t ==> pas
d'operation pour le rendre negatif ça
peut mal se passer
v.emplace_back(a,b); // shorter and
faster than pb(mp(a,b))
// 2D
vector< vector< bool > > myvector(
loula, vector<bool>( thenia, false )
);//ligne/col vérifié
vector.resize et non pas
vector.reserve
Rq: tab multidim plus rapide que
struct

std::reverse(copy.begin(),
copy.end());


int myArray[3][3];
for(auto& rows: myArray) // Iterating
over rows
{
for(auto& elem: rows)
{
// do some stuff
}
}
```

2. last nsi7a

```
Usually, we can expect the server to
execute about 10^8 instructions in a
second.
So, for a 1sec time limit,
N=10^6-10^7 : O(n) solution is
required.
N=10^5 : O(nlogn) solution.
N=10^4 : O(n^1.5 or n(logn)^2)
solution.
N=10^3 : O(n^2) solution.

_____
_____________
essaie des valeur limit surtout qd on
a tres peu de vars
|| l'ordre et la struc du code c'est
important dude -.-

3ess 3al prob el seeehliiin (!)
en mode strings 10^5, INTEGER number
ynajem ykoun >= ull

tu peut pas déclarer int
t[18][151072]; dans le main mais tu
peut le declarer en dehors du main
!!!!!!!!!!!!!!!
```

3. Math

```
~ <=> defini <=> !=-1
doubles can never give you more than
15 decimal digits of precision

//On x86, all types except 16-bit
(which is slow) are equally fast
base: 10^x
Maximum val for char: 2.10 u:2.41
size:1
Maximum val for short: 4.52 u:4.82
size:2
Maximum val for int: 9.33 u:9.63
size:4
Maximum val for ll: 18.96 u:19.27
size:8


#Syntaxe
#####################################
#########################
scanf: %x %o %e
setprecision() fait deja l'arrondi




Pi=2*acos(0)
//nbre de digit
(int) floor(1+ log10( (double)a) )
si on veux le nbre de case pour la
base 2
il faudra mettre log2
//racine n eme de a:
pow(a,1.0/n)
ne fait pas les a<0

nombre de diviseur en moyenne log(n) |
complexité gcd log(n)
ln(10^6) = 13.8 | EPS 1e-9
les func trigo prennent des val en
radian
```

```
#####################################
####################
Un coefficient binomial {{m} {n}}} est
divisible par un nombre premier p si
et seulement si
au moins un chiffre de n en base p est
plus grand que le chiffre
correspondant de m.


#####################################
#Get Divisors
#####################################
vector<ll> getdiv(ll g ){
set<ll> s;
ll q = (ll) sqrt(g);
for (ll i=1; i< q; ++i ){
if (g%i==0){
s.insert(i);
s.insert(g/i);
}
}
if (q*q==g) s.insert(q);
return vector<ll>(s.begin(),s.end() )
;
}


#####################################
####################
Geo
#####################################
########################
p est à l'interieur du triangle =>
l'aire des 3 ptit triange == aire du
triangle
eq droite 2D: ax + by+ c = 0
à partir de 2 point: (u,v) (u`,v`)

a = v`-v
b = u-u-u
c = -(bv+au)
intersection 2 droite (à verif)
ax + by+ c = 0
a`x + b`y+ c` = 0
y = (c+ c`*a/a`) / (b`*a/a`-b)
x = (-b*y-c)/a
(ne pas div par zero!)
```

4. Note

```
#PRQ : Plus rapide que
#PRQ* : difference significative >50ms
sur un bon nb de test
#####################################
########################
v operateur fois
111* Plus rapide que (long long)
int t[sz] PRQ long long t[sz]
boucle PRQ* memset (tab, val ,size);
double PRQ* long double mais moins
precis
It is a known fact than scanf() is
faster than cin
and getchar() is faster than scanf()
in general.
getchar_unlocked() is faster than
getchar(), hence fastest of all.
Similarly, there are getc_unlocked()
putc_unlocked(), and
putchar_unlocked() which are
nonthread-safe
versions of getc(), putc() and
putchar() respectively.
```

```
########## -- Attention 1.2
(10/03/2017)
#Err debile
#####################################
########
non init s->second (*.*)
attention a l input copié: 0 != O
vector< vector<int> > .clear(); NOO,
mais plutot: rep(i,n) vec[i].clear();
bel ka3ba bel ka3ba !! (err n'est pas
visible sur win,mais sur linux y'a
err)
n'oubli pas de dédoubler le test case
2_fois dans l'input et de voir.
```

```
ken kamalt jarabt el algo 3al les
exemple lel le5er rak maghlotech !
la brute force m'a bien aidé pour
avoir une vue global sur le prob
surtout qu'il y'avai une
formule a trouver.
ne9ess qlq %hell
dp [i][idx]
#team
#####################################
########################
ki fassertelhom fel le5er fehmouch
mli7. ça sert à rien en moin de 15mins
j'ai bien fait de laisser moez debug
mon prob A
il y avait un prob simple et on a raté
notre chance de le submit le plus vite
possible
```

5. rmq Range Min Q

```
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> vi;

class SegmentTree { // the segment tree is stored like a heap
array
private: vi st, A; // recall that vi is: typedef vector<int>
vi;
// Le A: le vecteur de base
// le st : segment tre arbre sous forme de vector
// We save Indexes !!!!!
int n;

int qwery(int p1, int p2)//ce sont des indexes !!!!
{
    return (A[p1] <= A[p2]) ? p1 : p2;
}
int left (int p) { return p << 1; } // same as binary heap
operations
int right(int p) { return (p << 1) + 1; }
void build(int p, int L, int R) { // O(n)
if (L == R) // as L == R, either one is fine
st[p] = L; // store the index
else { // recursively compute the values
build(left(p) , L , (L + R) / 2);
build(right(p), (L + R) / 2 + 1, R );
int p1 = st[left(p)], p2 = st[right(p)];
st[p] = qwery(p1, p2);
} }
int rmq(int p, int L, int R, int i, int j) { // O(log n)
if (i > R || j < L) return -1; // current segment outside
query range
if (L >= i && R <= j) return st[p]; // inside query range
// compute the min position in the left and right part of the
interval
int p1 = rmq(left(p) , L , (L+R) / 2, i, j);
int p2 = rmq(right(p), (L+R) / 2 + 1, R , i, j);
if (p1 == -1) return p2; // if we try to access segment
outside query
if (p2 == -1) return p1; // same as above
return qwery(p1, p2); // as in build routine
}
public:
SegmentTree(const vi &_A) {
A = _A; n = (int)A.size(); // copy content for local usage
st.assign(4 * n, 0); // create large enough vector of zeroes
build(1, 0, n - 1); // recursive build
}
int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); } //
overloading
};
int main() {
int arr[] = { 18, 17, 13, 19, 15, 11, 20 }; // the original
array
vi A(arr, arr + 7);
SegmentTree st(A);
printf("RMQ(1, 3) = %d\n", st.rmq(1, 3)); // answer = index 2
printf("RMQ(4, 6) = %d\n", st.rmq(4, 6)); // answer = index 5
} // return 0;
```

6. RSQ range sum q with update

```
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> vi;

class SegmentTree { // the segment tree is stored like a heap
array
public: vi st, A; // recall that vi is: typedef vector<int>
vi;
// Le A: le vecteur de base
// le st : segment tre arbre sous forme de vector: (!) seul
le st est 1-based le reste est 0 based
// We save Indexes !!!!!
int n;

int qwery(int p1, int p2)//ce sont des indexes !!!!
{
    return (A[p1] <= A[p2]) ? p1 : p2;
}
int left (int p) { return p << 1; } // same as binary heap
operations
int right(int p) { return (p << 1) + 1; }
void build(int p, int L, int R) { // O(n)
if (L == R) // as L == R, either one is fine
st[p] = L; // store the index
else { // recursively compute the values
build(left(p) , L , (L + R) / 2);
build(right(p), (L + R) / 2 + 1, R );
int p1 = st[left(p)], p2 = st[right(p)];
st[p] = qwery(p1, p2);
} }
int rmq(int p, int L, int R, int i, int j) { // O(log n)
if (i > R || j < L) return -1; // current segment outside
query range
```

```
if (L >= i && R <= j) return st[p]; // inside query range
// compute the min position in the left and right part of the
interval
int p1 = rmq(left(p) , L , (L+R) / 2, i, j);
int p2 = rmq(right(p), (L+R) / 2 + 1, R , i, j);
if (p1 == -1) return p2; // if we try to access segment
outside query
if (p2 == -1) return p1; // same as above
return qwery(p1, p2); // as in build routine
}
public:
SegmentTree(const vi &_A) {
A = _A; n = (int)A.size(); // copy content for local usage
st.assign(4 * n, 0); // create large enough vector of zeroes
build(1, 0, n - 1); // recursive build
}
int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); } //
overloading

private:
int update(int pos, int p, int value, int L, int R){ // O(log
n)
if (pos > R || pos < L) return st[p]; // current segment
outside query range
if (L == pos && R == pos) {
        cerr<<A[pos]<<"__";
    A[pos]=value;
        cerr<<A[pos]<<"__";
    return st[p];} // INDEXXX fel st!!! inside query range
// compute the min position in the left and right part of the
interval
int p1 = update(pos, left(p) ,value, L , (L+R) / 2);
int p2 = update(pos, right(p),value, (L+R) / 2 + 1, R);
// meme les segments outside query sont utilisé pour màj la
branche
return st[p]=qwery(p1, p2); // as in build routine
}
public:
void update(int pos, int value){
    //p=position dans st, pos=position dans A
    update(pos, 1, value, 0, n-1);
}


};
int main() {
int arr[] = { 18, 17, 13, 19, 15, 11, 20 }; // the original
array
vi A(arr, arr + 7);
SegmentTree st(A);
printf("RMQ(1, 3) = %d\n", st.rmq(1, 3)); // answer = index
2, zero based
printf("RMQ(4, 6) = %d\n", st.rmq(4, 6)); // answer = index 5
for(auto& e : st.A) cerr<<e<<" ";
cerr<<endl;
for(auto& e : st.st) cerr<<e<<" ";
st.update(1,5);// 0 - based
cerr<<endl;
for(auto& e : st.st) cerr<<e<<" ";
cerr<<endl;
for(auto& e : st.A) cerr<<e<<" ";
printf("RMQ(1, 3) = %d\n", st.rmq(1, 3)); // answer = index 1
} // return
```

7. String

```
std::ios_base::noskipws
char b[13];
string jj = string(&a[j]) ;
str = regex_replace(str,regex("("+jj.substr(0,1)+")"),bbb.substr(0,1) ) ;
int main()
{
std::bitset<16> b(5);
std::cout<<b.to_string();
std::cout<<std::endl;
b[0] = 0;
std::cout<<b.to_ulong();
return 0;
}
```