Team notebook

OBFUSCATION

December 12, 2019

Contents

1 Dynamic Programming

1.1 Binomial Coefficient

```
// Returns value of Binomial Coefficient C(n, k)
```

```
int binomialCoeff(int n, int k)
{
    int res = 1;
    if (k > n - k)
    {
        k = n - k;
    }
    // Calculate value of
    // [n * (n-1) *---* (n-k+1)] / [k * (k-1) *----* 1]
    for (int i = 0; i < k; ++i)
    {
        res *= (n - i);
        res /= (i + 1);
    }
    return res;
}
```

1.2 Edit Distance

```
int editDistDP(string str1, string str2, int m, int n)
{
    int dp[m + 1][n + 1];
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            if (i == 0)
                dp[i][j] = j; // Min. operations = j
            else if (j == 0)
                dp[i][j] = i; // Min. operations = i
            // If last characters are same, ignore last char
            // and recur for remaining string
            else if (str1[i - 1] == str2[j - 1])
                dp[i][j] = dp[i - 1][j - 1];
            // If the last character is different, consider all
            // possibilities and find the minimum
            else
                dp[i][j] = 1 + min(dp[i][j - 1],  // Insert
                                 dp[i - 1][j],    // Remove
                                 dp[i - 1][j - 1]); // Replace
        }
    }
    return dp[m][n];
}
```

1.3 Fibonacci

```
//direct
int fib(int n)
{
    double phi = (1 + sqrt(5)) / 2;
    return round(pow(phi, n) / sqrt(5));
}

//logn time and logn space

const int MAX = 1000;
int f[MAX] = {0};
int fib(int n)
{
    if (n == 0)
        return 0;
    if (n == 1 || n == 2)
        return (f[n] = 1);
    if (f[n] > 0)
        return f[n];

    int k = (n & 1) ? (n + 1) / 2 : n / 2;
    f[n] = (n & 1) ? (fib(k) * fib(k) + fib(k - 1) * fib(k - 1))
                   : (2 * fib(k - 1) + fib(k)) * fib(k);

    return f[n];
}
```

1.4 Knapsack

```
int knapSack(int w, int weight[], int value[], int n)
{
    int knap[n + 1][w + 1]; //dp table
    // Build table K[][] in bottom up manner
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= w; j++)
        {
            if (i == 0 || j == 0)
            {
                knap[i][j] = 0;
            }
            else if (weight[i - 1] <= j)
            {
                knap[i][j] = max(value[i - 1] + knap[i - 1][j -
                    weight[i - 1]], knap[i - 1][j]);
            }
            else
            {
                knap[i][j] = knap[i - 1][j];
            }
        }
    }
    return knap[n][w]; //answer
}
```

1.5 Longest Common Subsequence

```
/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs(string x, string y, int m, int n)
{
    int L[m + 1][n + 1]; //2D dp array
    /* Following steps build L[m+1][n+1] in bottom up fashion.
        Note
that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            if (i == 0 || j == 0) // base case
                L[i][j] = 0;
            else if (X[i - 1] == Y[j - 1]) //if two chars are
                equal
                L[i][j] = L[i - 1][j - 1] + 1;
            else //if two chars are not equal
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }
    /* L[m][n] contains length of LCS for X[0..n-1] and
        Y[0..m-1] */
    return L[m][n];
}
```

1.6 Longest Increasing Subsequence

```
//nlogn
int lis(int arr[], int N)
{
    int i;
    set<int> s;
    set<int>::iterator k;
    for (i = 0; i < N; i++)
    {
        if (s.insert(arr[i]).second)
        {
            k = s.find(arr[i]);

            k++;

            if (k != s.end())
                s.erase(k);
        }
    }
    // Note that set s may not contain actual LIS, but its size
        gives
    // us the length of LIS
    return s.size();
}
```

1.7 Minimum Coin Change

```
int minCoins(int coins[], int n)
{
    int table[n + 1];
    table[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        table[i] = INT_MAX; //initialize all values to Infinity
        // Go through all coins smaller than i
        for (int c : coins)
        {
            if (i - c >= 0)
            {
                table[i] = min(table[i], table[i - c] + 1);
            }
        }
    }
    return table[n];
}
```

1.8 Number of Coin Change

```
int coin(int S[], int m, int n)
{
    int table[n + 1];
    // Initialize all table values as 0
    memset(table, 0, sizeof(table));
    table[0] = 1;
    for (int i : S) //for each coin is S
        for (int j = i; j <= n; j++)
            table[j] += table[j - i];
    return table[n]; //answer
}
```

1.9 Rod Cutting

```
int cutRod(int price[], int n)
{
    int val[n + 1]; //dp table with optimal value for rod of
        length 0...n
    val[0] = 0;    //base case
    for (int i = 1; i <= n; i++)
    {
        int max_val = INT_MIN;
        for (int j = 0; j < i; j++)
        {
            max_val = max(max_val, price[j] + val[i - j - 1]);
        }
        val[i] = max_val; //answer for i in range 0...n
    }
    return val[n]; //answer
}
```

2 Graph

2.1 Bellman-Ford

```
// inside int main()
// initially, only source has distance = 0 and in the queue
//let source = s
int distance[n + 1] = {0}; // INIT TO INF
distance[s] = 0;
queue<int> q;
q.push(s);
bool in_queue[n + 1] = {0};
int cnt[n + 1] = {0};
in_queue[s] = 1;
bool negative = false;
while (!q.empty())
{
    int u = q.front();
    q.pop();
    in_queue[u] = 0;
    for (auto v : graph[u])
    {
        int x = v.first, weight = v.second;
        if (distance[u] + weight < distance[x])
        {                                    // if can relax
            distance[x] = distance[u] + weight; // relax
            if (!in_queue[x])
            {              // add to the queue
                q.push(x); // only if it is not already in the
                    queue
                cnt[x]++;
                in_queue[x] = 1;
                if (cnt[to] > n)
                {
                    negative = true;
                    break;
                }
            }
        }
    }
    if (negative)
    {
        break; //negative cycle
    }
}
```

2.2 BFS

```
queue<int> q;
bool visited[N];
int distance[N];
visited[x] = true;
distance[x] = 0;
q.push(x);
while (!q.empty())
{
    int s = q.front();
    q.pop();
    // process node s
    for (auto u : adj[s])
    {
        if (visited[u])
            continue;
        visited[u] = true;
        distance[u] = distance[s] + 1;
        q.push(u);
    }
}
```

2.3 Djikstra

```
queue<pair<int, int>> q;
for (int i = 1; i <= n; i++)
{
    distance[i] = INF;
}
distance[x] = 0;
q.push({0, x});
while (!q.empty())
{
    int a = q.top().second;
    q.pop();
    if (visited[a])
    {
        continue;
    }
    visited[a] = true;
    for (auto u : adj[a])
    {
        int b = u.first, w = u.second;
        if (distance[a] + w < distance[b])
        {
            distance[b] = distance[a] + w;
            q.push({-distance[b], b});
        }
    }
}
```

2.4 Floyd Warshall

```
//to init distance array
for (int i = 1; i <= n; i++)
{
    for (int j = 1; j <= n; j++)
    {
        if (i == j)
        {
            distance[i][j] = 0;
        }
        else if
        {
            (adj[i][j]) distance[i][j] = adj[i][j];
        }
        else
        {
            distance[i][j] = INF;
        }
    }
}
//to calculate values
for (int k = 1; k <= n; k++)
{
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            distance[i][j] = min(distance[i][j], distance[i][k] +
                distance[k][j]);
        }
    }
}
```

2.5 Iterative DFS

```
bool visited[n + 1] = {0};
stack<int> stack;
stack.push(s);

while (!stack.empty())
{
    int a = stack.top();
    stack.pop();

    if (!visited[a])
    {
        //process a
        visited[a] = true;
    }

    for (auto u : graph[a])
    {
        if (!visited[u])
        {
            stack.push(u);
        }
    }
}
```

3 Maths

3.1 Primality Test

```
bool MillerRabin(u64 n)
{ // returns true if n is prime, else returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0)
    {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37})
    {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

3.2 Prime Factorization

```
vector<int> factors(int n)
{
    vector<int> f;
    while (n % 2 == 0)
    {
        f.push_back(2);
        n = n / 2;
    }
    for (int i = 3; i <= sqrt(n); i = i + 2)
    {
        while (n % i == 0)
        {
            f.push_back(i);
            n = n / i;
        }
    }
    // This condition is to handle the case when n
    // is a prime number greater than 2
    if (n > 2)
        f.push_back(n);
    return f;
}
```

3.3 Sieve

```
int n = 20; //example
int sieve[n + 1];
memset(sieve, 0, sizeof(sieve)); //initialize to zero
for (int x = 2; x * x <= n; x++)
{
    if (sieve[x])
        continue;
    for (int u = x * x; u <= n; u += x)
    {
        sieve[u] = x;
    }
}
for (int i = 2; i <= n; i++)
{ //print all prime numbers
    if (!sieve[i])
        cout << i << "\n";
}
```

4 Misc

4.1 Bit Manipulation

```
n = n << i;          // Multiply n with 2^i
n = n >> 1;          // Divide n by 2^i
int computeXOR(int n) //compute xor from 1...n
{
    if (n % 4 == 0)
        return n;
    if (n % 4 == 1)
        return 1;
    if (n % 4 == 2)
        return n + 1;
    else
        return 0;
}
bool poweroftwo(int x) //check if x is power of two
{
    return x & (x - 1) == 0;
}
ch |= " ";          //Upper to Lower
ch &= "_";          //Lower to Upper
int logarithm(int x) //find log2
{
    int res = 0;
    while (x >>= 1)
        res++;
    return res;
}
//below all are 0 indexed
if (x & (1 << i))
{
    //ith bit is set
}
x |= (1 < < k)      //sets the kth bit of x to one
x &= ~(1 << k)   //unsets the kth bit of x to zero
x ^= (1 << k) //Inverts the kth bit of x
T = (S & (-S))   //T is a power of two with only one bit set
    which is the LSB.
~(x & 0)         //set all bits
S = (1 << n) - 1 //set all bits is s till n

// This function will return n % d.
// d must be one of: 1, 2, 4, 8, 16, 32, ...
unsigned int getModulo(unsigned int n, unsigned int d)
{
    return (n & (d - 1));
}
// __builtin_clz(x): the number of zeros at the beginning of the
    number
// __builtin_ctz(x) : the number of zeros at the end of the
    number
// __builtin_popcount(x) : the number of ones in the number
// __builtin_parity(x) : the parity(even or odd) of the number
    of ones
```

4.2 Generate Subsets

```
for (int b = 0; b < (1 << n); b++)
{
    vector<int> subset;
    for (int i = 0; i < n; i++)
    {
        if (b & (1 << i))
            subset.push_back(//ith element);
    }
}
```

4.3 Template

```
//OPTIMIZATIONS
#pragma GCC optimize("O3")
//(UNCOMMENT WHEN HAVING LOTS OF RECURSIONS)
//#pragma comment(linker, "/stack:200000000")
//(UNCOMMENT WHEN NEEDED)
//#pragma GCC
    optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
//#pragma GCC
    target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,tune=native")
//OPTIMIZATIONS
#include <bits/stdc++.h>
typedef long long ll;
typedef unsigned long long uu;
typedef long long int lll;
typedef unsigned long long int uuu;
using namespace std;
#define error(args...)                          \
    {                                           \
        string _s = #args;                      \
        replace(_s.begin(), _s.end(), ’,’, ’ ’); \
        stringstream _ss(_s);                   \
        istream_iterator<string> _it(_ss);      \
        err(_it, args);                         \
    }
```

```
void err(istream_iterator<string> it){}
template <typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args)
{
    cerr << *it << " = " << a << endl;
    err(++it, args...);
}
//use it as error(a,b,c);
#define cel(x, y) 1 + ((x - 1) / y)
const double PI = 3.141592653589793238463;
const int MOD = 1000000007;
const int INF = 0x3f3f3f3f;

/*
$ali103

URL: url

Solution Begins here
*/

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
}
```

4.4 Tricks

```
int ceilingdivision(int x, int y)
{
    return (x + y - 1) / y;
}
const double pi = 2 * acos(0.0) //accurate value of pi
```