ACM/ICPC CheatSheet

Puzzles

Contents

1 STL Useful Tips

1.1 Common libraries

```
/*** Functions ***/
#include<algorithm>
#include<functional> // for hash
#include<climits> // all useful constants
#include<cmath>
#include<cstdio>
#include<cstdlib> // random
```

```
#include<ctime>
#include<iostream>
#include<sstream>
/*** Data Structure ***/
#include<deque> // double ended queue
#include<list>
#include<queue> // including priority_queue
#include<stack>
#include<string>
#include<vector>
```

1.2 Useful constant

```
INT_MIN
INT_MAX
LONG_MIN
LONG_MAX
LLONG_MIN
LLONG_MAX
(~0u) // infinity (for long and long long)
      // use (~0u)>>2 for int.
```

1.3 Space waster

```
// consider to redefine data types to void data range problem
#define int long long // make everyone long long
#define double long double // make everyone long double

// function definitions

#undef int // main must return int
int main(void)
#define int long long // redefine int

// rest of program
```

1.4 Initialize array with predefined value

```
// for 1d array, use STL fill_n or fill to initialize array
fill(a, a+size_of_a, value)
fill_n(a, size_of_a, value)
// for 2d array, if want to fill in 0 or -1
memset(a, 0, sizeof(a));
  // otherwise, use a loop of fill or fill_n through every a[i]
  fill(a[i], a[i]+size_of_ai, value) // from 0 to number of row.
```

1.5 Modifying sequence operations

```
void copy(first, last, result);
void swap(a,b);
void swap(first1, last1, first2); // swap range
void replace(first, last, old_value, new_value); // replace in range
void replace_if(first, last, pred, new_value); // replace in conditions
  // pred can be represented in function
  // e.x. bool IsOdd (int i) { return ((i%2)==1); }
```

```
void reverse(first, last); // reverse a range of elements
void reverse_copy(first, last, result); // copy a reverse of range of elements
void random_shuffle(first, last); // using built-in random generator to shuffle array
```

1.6 Merge

```
// merge sorted ranges
void merge(first1, last1, first2, last2, result, comp);
// union of two sorted ranges
void set_union(first1, last1, first2, last2, result, comp);
// intersection of two sorted ranges
void set_interaction(first1, last1, first2, last2, result, comp);
// difference of two sorted ranges
void set_difference((first1, last1, first2, last2, result, comp);
```

1.7 String

```
// Searching
unsigned int find(const string &s2, unsigned int pos1 = 0);
unsigned int rfind(const string &s2, unsigned int pos1 = end);
unsigned int find_first_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_of(const string &s2, unsigned int pos1 = end);
unsigned int find_first_not_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_not_of(const string &s2, unsigned int pos1 = end);
// Insert, Erase, Replace
string& insert(unsigned int pos1, const string &s2);
string& insert(unsigned int pos1, unsigned int repetitions, char c);
string& erase(unsigned int pos = 0, unsigned int len = npos);
string& replace(unsigned int pos1, unsigned int len1, const string &s2);
string& replace(unsigned int pos1, unsigned int len1, unsigned int repetitions, char c);
// String streams
stringstream s1;
int i = 22;
s1 << "Hello world! " << i;
cout << s1.str() << endl;
```

1.8 Heap

```
template <class RandomAccessIterator>
  void push_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void push_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);

template <class RandomAccessIterator>
  void pop_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void pop_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);

template <class RandomAccessIterator>
  void make_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void make_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp );
```

```
template <class RandomAccessIterator>
  void sort_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void sort_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);
template <class RandomAccessIterator>
  RandomAccessIterator is_heap_until (RandomAccessIterator first,
                    RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  RandomAccessIterator is_heap_until (RandomAccessIterator first,
                    RandomAccessIterator last
                    Compare comp);
```

1.9 Sort

```
void sort(iterator first, iterator last);
void sort(iterator first, iterator last, LessThanFunction comp);
void stable_sort(iterator first, iterator last);
void stable_sort(iterator first, iterator last, LessThanFunction comp);
void partial_sort(iterator first, iterator middle, iterator last);
void partial_sort(iterator first, iterator middle, iterator last, LessThanFunction comp);
bool is_sorted(iterator first, iterator last);
bool is_sorted(iterator first, iterator last, LessThanOrEqualFunction comp);
// example for sort, if have array x, start_index, end_index;
sort(x+start_index, x+end_index);
```

1.10 Permutations

```
bool next_permutation(iterator first, iterator last);
bool next_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
bool prev_permutation(iterator first, iterator last);
bool prev_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
```

1.11 Searching

```
// will return address of iterator, call result as *iterator;
iterator find(iterator first, iterator last, const T &value);
iterator find_if(iterator first, iterator last, const T &value, TestFunction test);
bool binary_search(iterator first, iterator last, const T &value);
bool binary_search(iterator first, iterator last, const T &value, LessThanOrEqualFunction comp);
```

1.12 Random algorithm

```
srand(time(NULL));
// generate random numbers between [a,b)
rand() % (b - a) + a;
// generate random numbers between [0,b)
rand() % b;
// generate random permutations
random_permutation(anArray, anArray + 10);
random_permutation(aVector, aVector + 10);
```

2 Number Theory

2.1 Max or min

```
int max(int a, int b) { return a>b ? a:b; }
int min(int a, int b) { return a<b ? a:b; }
```

2.2 Greatest common divisor — GCD

```
int gcd(int a, int b)
{
  if (b==0) return a;
  else return gcd(b, a%b);
}
```

2.3 Least common multiple — LCM

```
int lcm(int a, int b)
{
  return a*b/gcd(a,b);
}
```

2.4 If prime number

```
bool prime(int n)
{
  if (n<2) return false;
  for (int i=2;i*i<=n;i++)
    if (n%i==0) return false;
  return true;
}
```

2.5 Leap year

```
bool isLeap(int n)
{
  if (n%100==0)
    if (n%400==0) return true;
    else return false;

  if (n%4==0) return true;
  else return false;
}
```

2.6 $a^b \bmod p$

```
long powmod(long base, long exp, long modulus) {
  base %= modulus;
  long result = 1;
  while (exp > 0) {
    if (exp & 1) result = (result * base) % modulus;
    base = (base * base) % modulus;
    exp >>= 1;
  }
```

```
  return result;
}
```

2.7 Factorial mod

```
//n! mod p
int factmod (int n, int p) {
  long long res = 1;
  while (n > 1) {
    res = (res * powmod (p-1, n/p, p)) % p;
    for (int i=2; i<=n%p; ++i)
      res=(res*i) %p;
    n /= p;
  }
  return int (res % p);
}
```

2.8 Generate combinations

```
// n>=m, choose M numbers from 1 to N.
void combination(int n, int m)
{
  if (n<m) return ;
  int a[50]={0};
  int k=0;

  for (int i=1;i<=m;i++) a[i]=i;
  while (true)
  {
    for (int i=1;i<=m;i++)
      cout << a[i] << " ";
    cout << endl;

    k=m;
    while ((k>0) && (n-a[k]==m-k)) k--;
    if (k==0) break;
    a[k]++;
    for (int i=k+1;i<=m;i++)
      a[i]=a[i-1]+1;
  }
}
```

3 Searching Algorithms

3.1 Find rank k in array

4 Dynamic Programming

4.1 Knapsack problems

4.2 Longest common subsequence

```
int dp[1001][1001];

int lcs(const string &s, const string &t)
{
```

```
  int m = s.size(), n = t.size();
  if (m == 0 || n == 0) return 0;
  for (int i=0; i<=m; ++i)
    dp[i][0] = 0;
  for (int j=1; j<=n; ++j)
    dp[0][j] = 0;
  for (int i=0; i<m; ++i)
    for (int j=0; j<n; ++j)
      if (s[i] == t[j])
            dp[i+1][j+1] = dp[i][j]+1;
          else
            dp[i+1][j+1] = max(dp[i+1][j], dp[i][j+1]);
  return dp[m][n];
}
```

4.3 Maximum submatrix

5 Trees

5.1 Tree representation in array

5.2 Tree traversal

6 Graph Theory

6.1 Graph representation

```
// The most common way to define graph is to use adjacency matrix
// example:
//     (1) (2) (3) (4) (5)
// (1)  2   0   5   0   0
// (2)  4   2   0   0   1
// (3)  3   0   0   1   4
// (4)  6   9   0   0   0
// (5)  1   1   1   1   5
// it's always a square matrix.
// suppose a graph has n nodes, if given exactly adjacency matrix
for (int i=1;i<=n;i++)
  for (int j=1;i<=n;j++)
  {
    cin << a[i][j] << endl;
  }
// Usually will go like this representation in data
// start_node end_node weight
// suppose m lines
for (int i=1;i<=m;i++)
{
  int x=0, y=0, t=0;
  cin >> x >> y >> t;
  a[x][y]=t;
  // if undirected graph
  a[y][x]=t;
}
// another variant: on the ith line, has data as
// end_node weight
// when you read data, you can assign matrix as
a[i][x]=t;
// if undirected graph
```

```
a[x][i]=t;

// Initialization of graph !!!IMPORTANT
// Depends on usage, normally initialize as 0 for all elements in matrix.
// so that 0 means no connection, non-0 means connection
// (for problem without weight, use weight as 1)
// If weights are important in this context (especially searching for path)
// Initialize graph as infinity for all elements in matrix.

// Another way to store graph is Adjacency list
// No space advantage if using array (unknown maximum number for in-degree).
// Big space advantage if using dynamic data structure (like list, vector).
// each row represent a node and its connectivity.
// we don't need it so much due to it's search efficiency.
// let's define a node as
struct Node{
  int id; // node id
  int w;  // weight
};
// suppose n nodes and m lines of inputs as
// start_node end_node weight
// assume using <vector> in this example
// g is a vector, and each element of g is also a vector of Node
for (int i=1;i<=m;i++)
{
  int x=0, y=0, t=0;
  cin >> x >> y >> t;
  Node temp; temp.id=y; temp.w=t;
  g[x].push_back(temp);
  // if undirected
  temp.id=x;
  g[y].push_back(temp);
}
// Note that you don't need this node structure if graph has only connectivity information.

/**** Special Structure ****/

// Special structure here is usually not a typical graph, like city-blocks, triangles
// They are represented in 2-d array and shows weights on nodes instead of edges.
// Note that in this case travel through edge has no cost, but visit node has cost.

// Triangles: Read data like this
// 1
// 1 2
// 4 2 7
// 7 3 1 5
// 6 2 9 4 6
for (int i=1;i<=n;i++)
  for (int j=i;j<=n;j++)
    cin >> a[i][j];

// Simple city-blocks: it's just like first form of adjacency matrix, but this time
// represents weights on nodes, may not be square matrix.
// 1 2 4 5 6
// 2 4 5 1 3
// 4 5 2 3 6
for (int i=1;i<=n;i++)
  for (int j=1;<=m;j++)
    cin >> a[i][j];
```

```
// More complex data structures: typical city-block structure may has some constraints on
// questions, but it has no boundaries. However, some questions requires to form a maze.
// In these cases, data structures can be very flexible, it totally depends on how the question
// presents the data. A usual way is to record it's adjacent blocks information:
struct Block{
  bool l[4]; // if has 8 neighbors then use bool l[8];
             // label them as your favor, e.x.
             //   1     1 2 3
             // 4 x 2   8 x 4
             //   3     7 6 5
             // true if there is path, false if there is boundary
  // other informations (optional)
  int weight;
  int component_id;
  // etc.
};

// Note that usually we use array from index 1 instead of 0 because sometimes
// you need index 0 as your boundary, and start from index 1 will give you
// advantage on locating nodes or positions
```

6.2 Flood fill algorithm

```
//component(i) denotes the
//component that node i is in
void flood_fill(new_component)
  do
    num_visited = 0
    for all nodes i
      if component(i) = -2
      num_visited = num_visited + 1
      component(i) = new_component

    for all neighbors j of node i
      if component(j) = nil
        component(j) = -2
  until num_visited = 0

void find_components()
  num_components = 0
  for all nodes i
    component(node i) = nil
  for all nodes i
    if component(node i) is nil
      num_components = num_components + 1
      component(i) = -2
      flood_fill(component num_components)
```

6.3 SPFA — shortest path

```
int q[3001]={0}; // queue for node
int d[1001]={0}; // record shortest path from start to ith node
bool f[1001]={0};
int a[1001][1001]={0}; // adjacency list
int w[1001][1001]={0}; // adjacency matrix
```

```
int main(void)
{
  int n=0, m=0;
  cin >> n >> m;

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0, z=0;
    cin >> x >> y >> z; // node x to node y has weight z
    a[x][0]++;
    a[x][a[x][0]]=y;
    w[x][y]=z;
    /*
    // for undirected graph
    a[x][0]++;
    a[y][a[y][0]]=x;
    w[y][x]=z;
    */
  }

  int s=0, e=0;
  cin >> s >> e; // s: start, e: end
  SPFA(s);
  cout << d[e] << endl;

  return 0;
}

void SPFA(int v0)
{
  int t,h,u,v;
  for (int i=0;i<1001;i++) d[i]=INT_MAX;
  for (int i=0;i<1001;i++) f[i]=false;
  d[v0]=0;
  h=0; t=1; q[1]=v0; f[v0]=true;

  while (h!=t)
  {
    h++;
    if (h>3000) h=1;
    u=q[h];
    for (int j=1; j<=a[u][0];j++)
    {
      v=a[u][j];
      if (d[u]+w[u][v]<d[v]) // change to > if calculating longest path
      {
        d[v]=d[u]+w[u][v];
        if (!f[v])
        {
          t++;
          if (t>3000) t=1;
          q[t]=v;
          f[v]=true;
        }
      }
    }
    f[u]=false;
  }
}
```

6.4 Floyd-Warshall algorithm – shortest path of all pairs

```
// map[i][j]=infinity at start
void floyd()
{
  for (int k=1; k<=n; k++)
    for (int i=1; i<=n; i++)
      for (int j=1; j<=n; j++)
        if (i!=j && j!=k && i!=k)
          if (map[i][k]+map[k][j]<map[i][j])
            map[i][j]=map[i][k]+map[k][j];
}
```

6.5 Prim — minimum spanning tree

```
int d[1001]={0};
bool v[1001]={0};
int a[1001][1001]={0};

int main(void)
{
  int n=0;
  cin >> n;
  for (int i=1;i<=n;i++)
  {
    int x=0, y=0, z=0;
    cin >> x >> y >> z;
    a[x][y]=z;
  }
  for (int i=1;i<=n;i++)
    for (int j=1;j<=n;j++)
      if (a[i][j]==0) a[i][j]=INT_MAX;

  cout << prim(1,n) << endl;
}
int prim(int u, int n)
{
  int mst=0,k;

  for (int i=0;i<d.length;i++) d[i]=INT_MAX;
  for (int i=0;i<v.length;i++) v[i]=false;

  d[u]=0;
  int i=u;

  while (i!=0)
  {
    v[i]=true;k=0;
    mst+=d[i];
    for (int j=1;j<=n;j++)
      if (!v[j])
      {
        if (a[i][j]<d[j]) d[j]=a[i][j];
        if (d[j]<d[k]) k=j;
      }
    i=k;
  }
  return mst;
```

```
}
```

6.6 Eulerian path

6.7 Topological sort