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>
#include<iomanip> // right justifying std::right and std::setw(width)
/*** Data Structure ***/
#include<deque> // double ended queue
#include<list>
#include<queue> // including priority_queue
#include<stack>
#include<string>
#include<vector>
```

1.2 I/O

```
// iostream and cstdio are both using I/O streams
// However, they have different behavior,
// pay attention on them if you're using them together.

// cin does not concern with '\n' at end of each line
// however scanf or getline does concern with '\n' at end of each line
// '\n' will be ignored when you use cin to read char.

// when you use getline(cin, str) to read a whole line of input
// please add an extra getline before inputing if previous inputs are numbers
cin >> n;
getline(cin, str) // wasted getline
getline(cin, str) // real input string
```

1.3 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.
```

1.4 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
```

1.5 Tricks in cmath

```
// when the number is too large. use powl instead of pow.
// will provide you more accuracy.
powl(a, b)
(int)round(p, (1.0/n)) // nth root of p
```

1.6 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.
```

1.7 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
```

1.8 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);
```

1.9 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;
```

1.10 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);
```

1.11 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);

/** sort a map **/
// You cannot directly sort a map<key type, mapped data type>
// if you only want to sort in key type
// you can use insert method to copy map into another map
// b.insert(make_pair(it->first, it->second) /* it is a map iterator */
// this will result a map which sorts key type in increasing order
// if you want to sort key type in decreasing order, then declare your map as
// something like:
// map<char, int, greater<char> >

// if you want to sort based on key, you need to copy the data to a vector
// where elements of vector are pair.
// you can define a PAIR type by using:
typedef pair<char, int> PAIR;

// suppose this is the map
map<char, int> a;

// sort vector in decreasing order
bool cmp_by_value(const PAIR& lhs, const PAIR& rhs) {
  return lhs.second > rhs.second;
}

// sort key in increasing order
bool cmp_by_char(const PAIR& lhs, const PAIR& rhs) {
  return lhs.first < rhs.first;
}

// copy map data to vector
vector<PAIR> b(a.begin(), a.end());

// sort data
sort(b.begin(), b.end(), cmp_by_value);

// you can still call your data by b[i].first and b[i].second.
// THE ABOVE CODES ARE EXAMPLE FOR SORTING A MAP.
// PLEASE USE IT FOR YOUR OWN DEMANDS.
```

1.12 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);
```

1.13 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);
```

1.14 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 Prime number under 100

```
// there are 25 numbers
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97
```

2.2 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; }
```

2.3 Greatest common divisor — GCD

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

2.4 Least common multiple — LCM

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

2.5 If prime number

```
bool prime(int n)
{
  if (n<2) return false;
```

```
  if (n<=3) return true;
  if (!(n%2) || !(n%3)) return false;
  for (int i=5;i*i<=n;i+=6)
    if (!(n%i) || !(n%(i+2))) return false;

  return true;
}
```

2.6 Prime factorization

```
// smallest prime factor of a number.
function factor(int n)
{
  int a;
  if (n%2==0)
    return 2;
  for (a=3;a<=sqrt(n);a++++)
  {
    if (n%a==0)
    return a;
  }
  return n;
}

// complete factorization
int r;
while (n>1)
{
    r = factor(n);
    printf("%d", r);
    n /= r;
}
```

2.7 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;
}
```

2.8 Binary exponiential

```
int binpow (int a, int n)
{
  int res = 1;
  while (n)
    if (n & 1)
    {
      res *= a;
      --n;
    }
```

```
    else
    {
      a *= a;
      n >>= 1;
    }
  return res;
}
```

2.9 $a^b \bmod p$

```
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;
}
```

2.10 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);
}
```

2.11 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;
  }
}
```

2.12 10-ary to m-ary

```
char a[16]={'0','1','2','3','4','5','6','7','8','9',
    'A','B','C','D','E','F'};

string tenToM(int n, int m)
{
  int temp=n;
  string result="";
  while (temp!=0)
  {
    result=a[temp%m]+result;
    temp/=m;
  }

  return result;
}
```

2.13 m-ary to 10-ary

```
string num="0123456789ABCDE";

int mToTen(string n, int m)
{
  int multi=1;
  int result=0;

  for (int i=n.size()-1;i>=0;i--)
  {
    result+=num.find(n[i])*multi;
    multi*=m;
  }

  return result;
}
```

2.14 Binomial coefficient

```
#define MAXN 100 // largest n or m
long binomial_coefficient(n,m) // compute n choose m
int n,m;
{
    int i,j;
    long bc[MAXN][MAXN];
    for (i=0; i<=n; i++) bc[i][0] = 1;
    for (j=0; j<=n; j++) bc[j][j] = 1;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
           bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
    return bc[n][m];
}
```

2.15 Catalan numbers

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{n}{k} \tag{1}$$

The first terms of this sequence are 2, 5, 14, 42, 132, 429, 1430 when $C_0 = 1$. This is the number of ways to build a balanced formula from n sets of left and right parentheses. It is also the number of triangulations of a convex polygon, the number of rooted binary tress on $n+1$ leaves and the number of paths across a lattice which do not rise above the main diagonal.

2.16 Eulerian numbers

$$\left\langle {n \atop k} \right\rangle = k \left\langle {n-1 \atop k} \right\rangle + (n-k+1) \left\langle {n-1 \atop k-1} \right\rangle \tag{2}$$

```
// This is the number of permutations of length n with exactly k ascending sequences or runs.
// Basis: k=0 has value 1
#define MAXN 100 // largest n or k
long eularian(n,k)
int n,m;
{
    int i,j;
    long e[MAXN][MAXN];
    for (i=0; i<=n; i++) e[i][0] = 1;
    for (j=0; j<=n; j++) e[0][j] = 0;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
           e[i][j] = k*e[i-1][j] + (i-j+1)*e[i-1][j-1];
    return e[n][k];
}
```

2.17 Karatsuba algorithm in Java

```
// fast algorithm to find multiplication of two big numbers.
import java.math.BigInteger;
import java.util.Random;

class Karatsuba {
  private final static BigInteger ZERO = new BigInteger("0");
  public static BigInteger karatsuba(BigInteger x, BigInteger y)
  {
    int N = Math.max(x.bitLength(), y.bitLength());
    if (N <= 2000) return x.multiply(y);
    N=(N/2)+(N %2);
    BigInteger b = x.shiftRight(N);
    BigInteger a = x.subtract(b.shiftLeft(N));
    BigInteger d = y.shiftRight(N);
    BigInteger c = y.subtract(d.shiftLeft(N));
    BigInteger ac = karatsuba(a, c);
    BigInteger bd = karatsuba(b, d);
    BigInteger abcd = karatsuba(a.add(b), c.add(d));
    return ac.add(abcd.subtract(ac).subtract(bd).shiftLeft(N)).add(bd.shiftLeft(2*N));
  }

  public static void main(String[] args)
  {
    long start, stop, elapsed;
    Random random = new Random();
    int N = Integer.parseInt(args[0]);
    BigInteger a = new BigInteger(N, random);
```

```
    BigInteger b = new BigInteger(N, random);
    start = System.currentTimeMillis();
    BigInteger c = karatsuba(a, b);
    stop = System.currentTimeMillis();
    System.out.println(stop - start);
    start = System.currentTimeMillis();
    BigInteger d = a.multiply(b);
    stop = System.currentTimeMillis();
    System.out.println(stop - start);
    System.out.println((c.equals(d)));
  }
}
```

2.18 Euler's totient function

```
// the positive integers less than or equal to n that are relatively prime to n.
int phi (int n)
{
  int result = n;
  for (int i=2; i*i<=n; ++i)
    if(n %i==0)
    {
      while(n %i==0)
        n /= i;
      result -= result / i;
    }
  if (n > 1)
    result -= result / n;
  return result;
}
```

2.19 Split plane

n lines can split a plane in $\frac{(n+1)n}{2} + 1$ sub-regions.

3 Searching Algorithms

3.1 Find rank k in array

```
int find(int l, int r, int k)
{
  int i=0,j=0,x=0,t=0;

  if (l==r) return a[l];
  x=a[(l+r)/2];
  t=a[x]; a[x]=a[r]; a[r]=t;
  i=l-1;

  for (int j=l; j<=r-1;j++)
    if (a[j]<=a[r])
    {
      i++;
      t=a[i]; a[i]=a[j]; a[j]=t;
    }
  i++;
  t=a[i]; a[i]=a[r]; a[r]=t;
  if (i==k) return a[i];
```

```
  if (i<k) return find(i+1, r,k);

  return find(l, i-1, k);
}
```

3.2 KMP Algorithm

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

typedef vector<int> VI;

void buildTable(string& w, VI& t)
{
  t = VI(w.length());
  int i = 2, j = 0;
  t[0] = -1; t[1] = 0;

  while(i < w.length())
  {
    if(w[i-1] == w[j]) { t[i] = j+1; i++; j++; }
    else if(j > 0) j = t[j];
    else { t[i] = 0; i++; }
  }
}

int KMP(string& s, string& w)
{
  int m = 0, i = 0;
  VI t;

  buildTable(w, t);
  while(m+i < s.length())
  {
    if(w[i] == s[m+i])
    {
      i++;
      if(i == w.length()) return m;
    }
    else
    {
      m += i-t[i];
      if(i > 0) i = t[i];
    }
  }
  return s.length();
}

int main(void)
{
  string a = (string) "The example above illustrates the general technique for assembling "+
    "the table with a minimum of fuss. The principle is that of the overall search: "+
    "most of the work was already done in getting to the current position, so very "+
    "little needs to be done in leaving it. The only minor complication is that the "+
    "logic which is correct late in the string erroneously gives non-proper "+
```

```
    "substrings at the beginning. This necessitates some initialization code.";

  string b = "table";

  int p = KMP(a, b);
  cout << p << ": " << a.substr(p, b.length()) << " " << b << endl;

  return 0;
}
```

4 Dynamic Programming

4.1 0/1 Knapsack problems

```
#include<iostream>

using namespace std;

int f[1000]={0};
int n=0, m=0;

int main(void)
{
  cin >> n >> m;

  for (int i=1;i<=n;i++)
  {
    int price=0, value=0;
    cin >> price >> value;

    for (int j=m;j>=price;j--)
      if (f[j-price]+value>f[j])
        f[j]=f[j-price]+value;
  }

  cout << f[m] << endl;

  return 0;
}
```

4.2 Complete Knapsack problems

```
#include<iostream>

using namespace std;

int f[1000]={0};
int n=0, m=0;

int main(void)
{
  cin >> n >> m;

  for (int i=1;i<=n;i++)
  {
    int price=0, value=0;
    cin >> price >> value;
```

```
    for (int j=price; j<=m; j++)
      if (f[j-price]+value>f[j])
        f[j]=f[j-price]+value;
  }

  cout << f[m] << endl;

  return 0;
}
```

4.3 Longest common subsequence (LCS)

```
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];
}
```

4.4 Longest increasing common sequence (LICS)

```
#include<iostream>

using namespace std;

int a[100]={0};
int b[100]={0};
int f[100]={0};
int n=0, m=0;

int main(void)
{
  cin >> n;
  for (int i=1;i<=n;i++) cin >> a[i];
  cin >> m;
  for (int i=1;i<=m;i++) cin >> b[i];

  for (int i=1;i<=n;i++)
  {
    int k=0;
    for (int j=1;j<=m;j++)
    {
      if (a[i]>b[j] && f[j]>k) k=f[j];
      else if (a[i]==b[j] && k+1>f[j]) f[j]=k+1;
```

```
    }
  }

  int ans=0;
  for (int i=1;i<=m;i++)
    if (f[i]>ans) ans=f[i];

  cout << ans << endl;

  return 0;
}
```

4.5 Longest Increasing Subsequence (LIS)

```
#include<iostream>

using namespace std;

int n=0;
int a[100]={0}, f[100]={0}, x[100]={0};

int main(void)
{
  cin >> n;
  for (int i=1;i<=n;i++)
  {
    cin >> a[i];
    x[i]=INT_MAX;
  }

  f[0]=0;

  int ans=0;
  for(int i=1;i<=n;i++)
  {
    int l=0, r=i;

    while (l+1<r)
    {
      int m=(l+r)/2;
      if (x[m]<a[i]) l=m; else r=m;
      // change to x[m]<=a[i] for non-decreasing case
    }

    f[i]=l+1;
    x[l+1]=a[i];
    if (f[i]>ans) ans=f[i];
  }

  cout << ans << endl;

  return 0;
}
```

4.6 Maximum submatrix

```
// URAL 1146 Maximum Sum
#include<iostream>

using namespace std;

int a[150][150]={0};
int c[200]={0};

int maxarray(int n)
{
   int b=0, sum=-100000000;
   for (int i=1;i<=n;i++)
   {
      if (b>0) b+=c[i];
      else b=c[i];
      if (b>sum) sum=b;
   }

   return sum;
}

int maxmatrix(int n)
{
   int sum=-100000000, max=0;

   for (int i=1;i<=n;i++)
   {
      for (int j=1;j<=n;j++)
         c[j]=0;

      for (int j=i;j<=n;j++)
      {
         for (int k=1;k<=n;k++)
            c[k]+=a[j][k];
         max=maxarray(n);
         if (max>sum) sum=max;
      }
   }

   return sum;
}

int main(void)
{
   int n=0;
   cin >> n;
   for (int i=1;i<=n;i++)
      for (int j=1;j<=n;j++)
         cin >> a[i][j];

   cout << maxmatrix(n);
   return 0;
}
```

4.7 Partitions of integers

```
#define MAXN 100 // largest n or m
long int_coefficient(n,k) // compute f(n,k)
```

```
int n,m;
{
    int i,j;
    long f[[MAXN][MAXN];
    f [1][1] = 1;
    for (i=0;i<=n;i++) f[i][0] = 0;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
           if (i-j <= 0)
               f[i][j] = f[i][k-1];
           else
               f[i][j] = f[i-j][k]+f[i][k-1];
    return f[n][k];
}
```

4.8 Partitions of sets

Number of ways to partition $n+1$ items into k sets.

$$\left\{ {n \atop k} \right\} = k \left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\} \tag{3}$$

where

$$\left\{ {n \atop 1} \right\} = \left\{ {n \atop n} \right\} = 1 \tag{4}$$

5 Trees

5.1 Tree traversal

```
int L[100]={0};
int R[100]={0};

void DLR(int m)
{
  cout << m << " ";
  if (L[m]!=0) DLR(L[m]);
  if (R[m]!=0) DLR(R[m]);
}

void LDR(int m)
{
  if (L[m]!=0) LDR(L[m]);
  cout << m << " ";
  if (R[m]!=0) LDR(R[m]);
}

void LRD(int m)
{
  if (L[m]!=0) LRD(L[m]);
  if (R[m]!=0) LRD(R[m]);
  cout << m << " ";
}

int main(void)
{
  cin >> n;
  for (int i=1;i<=n;i++)
    cin >> L[i] >> R[i];
```

```
  DLR(1); cout << endl;
  LDR(1); cout << endl;
  LRD(1); cout << endl;

  return 0;
}
```

5.2 Depth and width of tree

```
#include <iostream>
#include <queue>
#include <stack>

using namespace std;

int l[100]={0};
int r[100]={0};
stack<int> mystack;
int n=0;
int w=0;
int d=0;

int depth(int n)
{
  if (l[n]==0 && r[n]==0)
    return 1;

  int depthl=depth(l[n]);
  int depthr=depth(r[n]);
  int dep=depthl>depthr ? depthl:depthr;
  return dep+1;
}

void width(int n)
{
  if (n<=d)
  {
    int t=0,x;
    stack<int> tmpstack;
    while (!mystack.empty())
    {
      x=mystack.top();
      mystack.pop();
      if (x!=0)
      {
        t++;
        tmpstack.push(l[x]);
        tmpstack.push(r[x]);
      }
    }

    w=w>t?w:t;
    mystack=tmpstack;
    width(n+1);
  }
}
```

```
int main(void)
{
  cin >> n;

  for (int i=1;i<=n;i++)
    cin >> l[i] >> r[i];

  d=depth(1);
  mystack.push(1);
  width(1);

  cout << w << " " << d << endl;

  return 0;
}
```

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 circuit

```cpp
// USACO Fence
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0, c=0;

void dfs(int k)
{
  for (int i=1;i<=n;i++)
    if (g[k][i])
    {
      g[k][i]=false;
      g[i][k]=false;
      dfs(i);
    }
  m++;
  ans[m]=k;
}

int main(void)
{
  cin >> n >> m;

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
    g[x][y]=true;
    g[y][x]=true;
    f[x]++;
    f[y]++;
  }

  m=0;
  int k1=0;
  for (int i=1;i<=n;i++)
  {
    if (f[i]%2==1) k1++;
    if (k1>2)
    {
      cout << "error" << endl;
      return 0;
    }
    if (f[i]%2 && c==0) c=i;
  }

  if (c==0) c=1;
  dfs(x);

  for (int i=m;i>=1;i--) cout << ans[i] << endl;
  return 0;
}
```

6.7 Topological sort

```cpp
// Find any solution of topological sort.
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0;

void dfs(int k)
{
  int i=0;
  v[k]=true;
  for (int i=1;i<=n;i++)
    if (g[k][i] && !v[i]) dfs(i);

  m++;
  ans[m]=k;
}

int main(void)
{
  cin >> n >> m;

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
    cin >> x >> y;
    g[y][x]=true;
  }

  m=0;
  for (int i=1;i<=n;i++)
    if (!v[i]) dfs(i);

  for (int i=1;i<=n;i++) cout << ans[i] << endl;
  return 0;
}
```

```cpp
// Find the order of topological sort is dictionary minimum
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0;

int main(void)
{
  cin >> n >> m;

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
    cin >> x >> y;
    g[x][y]=true;
    f[y]++;
  }
```

```
  for (int i=1;i<=n;i++)
  {
    for (int j=1;j<=n;j++)
    {
      if (f[j]==0 && !v[j]) break;

      if (f[j]!=0)
      {
        cout << "error" << endl;
        return 0;
      }

      ans[i]=j;
      v[j]=true;
      for (int k=1;k<=n;k++)
        if (g[j][k]) f[k]--;
    }
  }
  for (int i=1;i<=n;i++) cout << ans[i] << endl;
  return 0;
}
```