Contents

1 Basic
- 1.1 Syntax . . . 1
- 1.2 Linux Command . . . 1
- 1.3 Substring . . . 1
- 1.4 BigInteger . . . 2

2 Data Structure
- 2.1 Tree . . . 2
- 2.2 Disjoint Set . . . 2
- 2.3 Segment Tree . . . 2
- 2.4 Tree Policy . . . 3
- 2.5 KMP . . . 3
- 2.6 LCA . . . 3

3 Divide and Conquer
- 3.1 MaximumSubArray . . . 4
- 3.2 Closet Set Pair . . . 4

4 Dynamic Programming
- 4.1 LCS . . . 4
- 4.2 LIS . . . 5
- 4.3 Knapsack . . . 5
- 4.4 ChangeCoin . . . 5
- 4.5 String Edition . . . 5
- 4.6 Chain Matrix Mul . . . 6

5 Search
- 5.1 Binary Search . . . 6

6 Sequence
- 6.1 RSQ(Prefix Sum) . . . 7
- 6.2 RSQ(2DPrefix Sum) . . . 7
- 6.3 RSQ(Fenwick Tree) . . . 7

7 Sorting
- 7.1 Counting Sort . . . 8
- 7.2 Topology Sort . . . 8
- 7.3 Topology Sort with DFS(check 有無環) . . . 8

8 Graph
- 8.1 DFS I . . . 9
- 8.2 DFS II . . . 9
- 8.3 DFS Tree . . . 9
- 8.4 BFS . . . 9
- 8.5 AOE . . . 9
- 8.6 Dijkstra . . . 10
- 8.7 SPFA . . . 11
- 8.8 BellmanFord . . . 11
- 8.9 FloydWarshall . . . 11
- 8.10 Kruskal . . . 12
- 8.11 Articulation Point . . . 12
- 8.12 KM . . . 13
- 8.13 Bipartite Matching . . . 14
- 8.14 Bipartite Check1 . . . 14
- 8.15 CLE Directed MST . . . 15
- 8.16 Dinic . . . 15
- 8.17 MCMF . . . 16

9 Number
- 9.1 Sieve . . . 17
- 9.2 Power . . . 17
- 9.3 Euler . . . 17
- 9.4 Factors . . . 17
- 9.5 Extend Euclidean . . . 17
- 9.6 Matrix . . . 17
- 9.7 GaussElimination . . . 18

10 Geometry
- 10.1 Geometry . . . 18
- 10.2 Lines Intersection1 . . . 18
- 10.3 Lines Intersection2 . . . 19
- 10.4 Polygon Inside Or Outside . . . 19
- 10.5 Convex Hull . . . 20

1 Basic

1.1 Syntax

```cpp
// 加速cin, cout
#define IOS cin.tie(nullptr); cout.tie(nullptr);
    ios_base::sync_with_stdio(false);
int main(int argc, char const *argv[])
{
  // String to Integer
  char str[30] = {'-', '1', '2', '3', '4', '5', '\0'};
  printf("%d\n", stoi(str));
  // Integer to String
  int x = 185;
  char temp[30];
  int base = 10;
  itoa(x, temp, base);
  printf("%s\n", temp);
  // String to Double
  char strd[30] = {'0', '.', '6', '0', '2', '9', '\0'};
  printf("%lf\n", stod(strd));
  // Double to String
  double y = 3.1415926;
  string dstr = to_string(y);
  cout << dstr << endl;
  // String initialize
  char null[30] = {'\0'};
  char A[30];
  strcpy(A, null);
  // String Length
  char strl[30] = {'H', 'E', 'L', 'L', 'O', '\0'};
  printf("%d\n", strlen(strl));
  // String Reverse
  char a[] = {'a', 'b', 'c', 'd', 'e', 'f', '\0'};
  strrev(a); reverse(a, a + 6);
  string s = "abcdefg";
  reverse(s.begin(), s.end());
  /* Complexity
  O(N) 大概 N 可以到 1億
  O(N log N) 大概 N 可以到數百萬~千萬
  O(N^1.5) 大概可以到數萬
  O(N^2) 大概 5000~10000
   */
  return 0;
}
```

1.2 Linux Command

```
1.創一個.in檔案
touch PA.in
2.執行exe檔案
./PA.exe > PA.in < PA.out
3.打開.out或.in檔
cat PA.in
4.比較答案
diff PA.out ans.txt
```

1.3 Substring

```cpp
#include <bits/stdc++.h>
using namespace std;
bool isSubstring(string a, string b){
        bool is =0;
        if(b.find(a) != std::string::npos){
            is =1;
        }
        return is;
}
//check if string a is substring of b
int main(){
    string a = "123",b = "12345";
    // "123" 是不是 substring "12345"
    if(isSubstring(a,b)) cout << "yes"<<endl;
  else cout << "no"<<endl;
    return 0;
}
```

1.4 BigInteger

```java
import java.math.BigInteger;
import java.util.Scanner;
class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        BigInteger n = input.nextBigInteger();
        BigInteger m = input.nextBigInteger();
        n.add(m); a.subtract(m); n.multiply(m); n.
            divide(m); n.mod(m);
        n.pow(m.intValue()); n.gcd(m); n.negate(); n.
            abs();
    }
}
```

2 Data Structure

2.1 Tree

```cpp
struct node
{
  int data;
  node* left;
  node* right;
};

void preOrder(node *root) {

    cout << root->data << " ";
    if(root->left != NULL)  preOrder(root->left);
    if(root->right != NULL) preOrder(root->right);
}

void postOrder(node *root) {

    if(root->left != NULL)  postOrder(root->left);
    if(root->right != NULL) postOrder(root->right);
    cout << root->data << " ";
}
void inOrder(node *root) {

    if(root->left != NULL) inOrder(root->left);
    cout << root->data << " ";
    if(root->right != NULL) inOrder(root->right);
}
```

2.2 Disjoint Set

```cpp
const int n = 6; // number of nodes
int p[n+10];
void init()
{
  for(int i = 0; i < n; i ++){
    p[i] = -1;
  }
}
int find(int x){
  int root, trail, lead;
  for (root = x; p[root] >= 0; root = p[root]);
  for (trail = x; trail != root; trail = lead) {
        lead = p[trail];
        p[trail]= root;
  }
  return root;
}
void uni(int x ,int y)
{
  int xRoot = find(x), yRoot = find(y);
  if(xRoot != yRoot){
    if(p[xRoot] < p[yRoot]){
      p[xRoot] += p[yRoot];
      p[yRoot] = xRoot;
    }
    else{
      p[yRoot] += p[xRoot];
      p[xRoot] = yRoot;
    }
  }
}
```

2.3 Segment Tree

```cpp
#include <bits/stdc++.h>
using namespace std;
const int n = 8;
int B[n] = {18, 17, 13, 19, 15, 11, 20, 87};
typedef vector<int> vi;
vi A (B, B + 8);
vi ST;
void ST_Build(vi &ST, const vi &A, int vertex, int L,
    int R)
{
  if(L == R) ST[vertex] = L;
  else
  {
    int nL = vertex * 2, nR = vertex * 2 + 1;
    ST_Build(ST, A, nL, L, L + (R - L) / 2);
    ST_Build(ST, A, nR, L + (R - L) / 2 + 1, R);
    int indexL = ST[nL], indexR = ST[nR];
    int valueL = A[indexL], valueR = A[indexR];
    ST[vertex] = valueL <= valueR ? indexL : indexR;
  }
}

void ST_Creation(vi &ST, const vi &A)
{
  int len = 4 * A.size();
  ST.assign(len, 0);
  ST_Build(ST, A, 1, 0, A.size()-1);
}
int query(vi &ST, const vi &A, int vertex, int L, int R
    , int qL, int qR)
{
  int temp, mid = (L + R) / 2;
  if(qL <= L && R <= qR) return ST[vertex];
  if(qR <= mid)
  { //all we want at the left child
    return query(ST, A, vertex * 2, L, mid, qL, qR);
  }
  if(qL > mid)
  { // all we want at the right child
    return query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
        qR);
  }
  return A[query(ST, A, vertex * 2, L, mid, qL, qR)] <=
      A[query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
      qR)]
      ? query(ST, A, vertex * 2, L, mid, qL, qR) :
          query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
          qR);

}
void update(vi &ST, vi &A, int x, int L, int R, int p,
    int v)
{
  // p is the index where you want to update
  // v is the value will be update in A[p];
  int mid = L + (R - L) / 2;
  if(L == R) A[ST[x]] = v;
  else
  {
    if(p <= mid) update(ST, A, x*2, L, mid, p, v);
    else update(ST, A, x*2+1, mid+1, R, p, v);
    ST[x] = (A[ST[x*2]] <= A[ST[x*2+1]]) ? ST[x*2] : ST
        [x*2+1];
  }
}
```

```
int main(int argc, char const *argv[])
{
  ST_Creation(ST, A);
  printf("%d\n", query(ST, A, 1, 0, n-1, 3, 7));
  // query return the index
  printf("%d\n", A[query(ST, A, 1, 0, n-1, 3, 7)]);
  update(ST, A, 1, 0, n-1, 5, 18);
  // query and update first to fifth parameter dont change
  // ST, A, 1, 0, n-1
  // last two would be
  // query: the range(array index) you want to query
  // update: fisrt is the index you want to update, second is the value will be
  return 0;
}
```

2.4 Tree Policy

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp>
#include <functional> // for less
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> new_data_set;
new_data_set t;
int main()
{
    t.insert(5);
    t.insert(6);
    t.insert(3);
    t.insert(1);
    // the smallest is (0), bigest is (n-1), kth small is (k-1)
    int num = *t.find_by_order(0);
    printf("%d\n", num); // print 1
    num = *t.find_by_order(t.size()-1);
    printf("%d\n", num); // print 6
    // find the index
    int index = t.order_of_key(6);
    printf("%d\n", index); // print 3
    // cheak if there exist x
    int x = 5;
    int check = t.erase(x);
    if(check == 0) printf("t not contain 5\n");
    else if(check == 1) printf("t conain 5\n");
    //tree policy like set
    t.insert(5); t.insert(5);
    // get the size of t
    printf("%d\n", t.size()); // print 4
    return 0;
}
```

2.5 KMP

```
int kmp(string text, string pattern){
  if(pattern.size()==0) return -1;
  int patLen=pattern.size();
  int textLen=text.size();
  int LPS[patLen]={0};
  int i=1,j=0;

  while(i<patLen){
    if(pattern[i]==pattern[j]){
      LPS[i++]=++j;
    }
    else{
      if(j) j=LPS[j-1];
      else LPS[i++]=0;
    }
  }
  i=j=0;
  while(i<textLen){
    if(pattern[j]==text[i]){
      i++;j++;
    }
    if(j==patLen) return i-j;
    else{
      if(i<textLen && pattern[j]!=text[i]){
        if(j) j=LPS[j-1];
        else i++;
      }
    }
  }
  return -1;
}

int main(){
  string text,pattern;
  getline(cin, text);
  getline(cin, pattern);
  int index=kmp(text,pattern);
  if(index>0){
    cout<<"\nPattern found at : "<<index<<"\n";
  }
  else{
    cout<<"\nPattern not found!\n";
  }
}
```

2.6 LCA

```
#define max 100
#define lg_max 7
vector<int> graph[max];
int parent[max][lg_max], level[max], lg[max];
int n;
void log()
{
    for (int i = 2; i < max; i++)
        lg[i] = lg[i / 2] + 1;
}
void dfs(int u, int p)
{
    for (auto v : graph[u]){
        if (v != p){
            level[v] = level[u] + 1;
            parent[v][0] = u;
            dfs(v, u);
        }
    }
}
void build()
{
    for (int j = 1; j <= lg[n]; j++)
    {
        for (int i = 1; i <= n; i++)
        {
            parent[i][j] = parent[parent[i][j - 1]][j - 1];
        }
    }
}
int lca(int u, int v)
{
    if (level[u] < level[v]) return lca(v, u);
    for (int i = lg[n]; i >= 0; i--){
        if (level[u] - (1 << i) >= level[v]){
            u = parent[u][i];
        }
    }
    if (u == v) return u;
    for (int i = lg[n]; i >= 0; i--){
        if (parent[u][i] != parent[v][i]){
            u = parent[u][i];
            v = parent[v][i];
        }
```

```
    }
    return parent[u][0];
}
int main()
{
    log();
    int x, y;
    scanf("%d", &n);
    for (int i = 0; i < n - 1; i++){
        scanf("%d%d", &x, &y);
        graph[x].push_back(y);
        graph[y].push_back(x);
    }
    dfs(1, 1);
    build();
    scanf("%d%d", &x, &y);
    printf("%d\n", lca(x, y));
}
```

3 Divide and Conquer

3.1 MaximumSubArray

```
#include <bits/stdc++.h>
using namespace std;
const int n = 16;
int arr[n] = {13, -3, -25, 20, -3, -16, -23,
        18, 20, -7, 12, -5, -22, 15, -4, 7};

int findMaxCrosing(int left, int mid, int right){
  int maxl = 0x80000000;
  int sum = 0;
  for(int i = mid; i >= left; i--){
    sum += arr[i];
    if(sum > maxl) maxl = sum;
  }
  int maxr = 0x80000000;
  sum = 0;
  for(int i = mid + 1; i <= right; i++){
    sum += arr[i];
    if(sum > maxr) maxr = sum;
  }

  return (maxl + maxr);
}

int findMaxSub(int left, int right)
{
  if(left == right){
    return arr[left];
  }
  else{
    int mid = left + (right - left) / 2;
    int maxl = findMaxSub(left, mid);
    int maxr = findMaxSub(mid + 1, right);
    int res = max(maxl, maxr);
    res = max(res, findMaxCrosing(left, mid, right));
    return res;
  }
}


int main(int argc, char const *argv[])
{
  printf("%d\n", findMaxSub(0, n-1));
  return 0;
}
```

3.2 Closet Set Pair

```
struct point2D
{
  double x, y;
  bool operator< (point2D const other) const{
    return x < other.x;
  }
  bool operator> (point2D const other) const{
    return y > other.y;
  }
};
point2D p[10000+10];

double dis(point2D p1, point2D p2)
{
  return sqrt(((p1.x - p2.x) * (p1.x - p2.x)) + ((p1.y
      - p2.y) * (p1.y - p2.y)));
}
double bruteforce(int start, int n){
  double mind = 2e9;
  for(int i = start; i < n - 1; i++){
    for(int j = i + 1; j < n; j++){
      mind = min(mind, dis(p[i], p[j]));
    }
  }
  return mind;
}
double findcp(int left, int right,int n)
{
  if(n <= 3){
    return bruteforce(left, n);
  }
  double mind;
  int mid = left + (right - left) / 2;
  double cl = findcp(left, mid, mid - left + 1);
  double cr = findcp(mid + 1, right, right - mid);
  mind = min(cl, cr);
  vector<point2D> v;
  for(int i = left; i <= right; i++){
    if(p[i].x <= p[mid].x + mind && p[i].x >= p[mid].x
        - mind)
      v.push_back(p[i]);
  }
  sort(v.begin(), v.end(), greater<point2D>());

  for(vector<point2D>::iterator it = v.begin(); it != v
      .end()-1; it++){
    for(vector<point2D>::iterator jt = it + 1; jt != v.
        end(); jt++){
      mind = min(mind, dis(*it, *jt));
    }
  }
  return mind;
}

int main(int argc, char const *argv[])
{
  int n;
  double min;
  while(cin >> n && n)
  {
    for(int i = 0; i < n; i++){
      cin >> p[i].x >> p[i].y;
    }
    sort(p, p + n);
    min = findcp(0, n-1, n);
    if(min < 10000) printf("%.4lf\n", min);
    else printf("INFINITY\n");

  }
  return 0;
}
```

4 Dynamic Programming

4.1 LCS

```cpp
const int maxn = 10000; // maxn is maximum length of arrp and arrq
int arrp[maxn], arrq[maxn];
int dp[maxn+5][maxn+5];
int p, q; // p is the length of arrp, q is the length of arrq
void LCS()
{
  memset(dp, 0, sizeof(dp));

  for(int i = 1; i <= p; i++){
    for(int j = 1; j <= q; j++){
      if(arrp[i] == arrq[j]){
        dp[i][j] = 1 + dp[i-1][j-1];
      }
      else{
        dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
      }
    }
  }
  // dp[p][q] is the answer
}
```

4.2 LIS

```cpp
int LIS(vector<int>& s)
{
    if (s.size() == 0) return 0;

    vector<int> v;
    v.push_back(s[0]);

    for (int i = 1; i < s.size(); ++i)
    {
        int n = s[i];

        if (n > v.back())
            v.push_back(n);
        else
            *lower_bound(v.begin(), v.end(), n) = n;
    }

    return v.size();
}
```

4.3 Knapsack

```cpp
#include<bits/stdc++.h>
using namespace std;
int dp[1005][1005];
int track[1005][1005];
struct Item{
  int value, weight;
};
vector <Item> item;
int main(){
  int n, W, t, t1, t2, temp_w, temp_v;
  vector <int> ans_item;
  cin >> n;
  while(n--){
    cin >> W >> t;//W = total weight, t = # of item
    item.clear(); ans_item.clear();
    item.push_back(Item{0, 0});
    for(int i = 0 ; i<t ; i++){
      cin >> t1 >> t2;
      item.push_back(Item{t1, t2});
    }
    memset(track, 0, sizeof(track));
    for(int i = 0 ; i<=t ; i++){//row - i
      temp_w = item[i].weight;
      temp_v = item[i].value;
      for(int w = 0 ; w<=W ; w++){
        if(i == 0 || w == 0){
          dp[w][i] = 0;
          continue;
        }
        if(temp_w <= w){
          //dp[w][i] = max(dp[w][i-1], dp[w - temp_w][i-1] + temp_v);
          if((dp[w - temp_w][i-1] + temp_v) > dp[w][i-1]){
            dp[w][i] = dp[w - temp_w][i-1] + temp_v;
            track[w][i] = true;//true=有放
          }
          else{
            dp[w][i] = dp[w][i-1];
          }
        }
        else{
          dp[w][i] = dp[w][i-1];
        }
      }
    }
    cout << dp[W][t] << endl;
    //backtracking
    int ii = t-1, ww = W;
    while(ii != 0){
      if(track[ww][ii]){
        ww -= item[ii].weight;
        ans_item.push_back(ii);
      }
      ii -= 1;
    }
  }
}
```

4.4 ChangeCoin

```cpp
#include<bits/stdc++.h>
using namespace std;
long long dp[30005][5];
int cents[5] = {1, 5, 10, 25, 50};
int main(){
  long long N, ans;
  while(cin >> N){
    for(int k = 0 ; k<5 ; k++){
      for(int n = 0 ; n<=N ; n++){
        if(k == 0 || n == 0){
          dp[n][k] = 1;
          continue;
        }
        if(n < cents[k]){
          dp[n][k] = dp[n][k-1];
        }
        else dp[n][k] = dp[n][k-1] + dp[n - cents[k]][k];
      }
    }
    ans = dp[N][4];
    printf("There are %lld ways to produce %lld cents change.\n", ans, N);

  }
}
```

4.5 String Edition

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 90;
char s1[maxn], s2[maxn];
int dp[maxn][maxn];
struct Coor
{
  int x, y;
};
```

```
Coor backtracking[maxn][maxn];
vector<Coor>ans;
int main(int argc, char const *argv[])
{
  bool begining = true;
  while(gets(s1)){
    gets(s2);
    if(begining) begining = false;
    else printf("\n");
    memset(dp, 0, sizeof(dp));
    memset(backtracking, 0, sizeof(backtracking));
    ans.clear();
    for(int i = 1; i <= strlen(s2); i++) {
      dp[0][i] = dp[0][i-1] + 1;
      backtracking[0][i].x = 0;
      backtracking[0][i].y = i-1;
    }
    for(int i = 1; i <= strlen(s1); i++) {
      dp[i][0] = dp[i-1][0] + 1;
      backtracking[i][0].x = i-1;
      backtracking[i][0].y = 0;
    }
    for(int i = 1; i <= strlen(s1); i++){
      for(int j = 1; j <= strlen(s2); j++){
        if(s1[i-1] == s2[j-1]) {
          dp[i][j] = dp[i-1][j-1];
          backtracking[i][j] = Coor{i-1, j-1};
        }
        else{
          dp[i][j] = min(dp[i][j-1], min(dp[i-1][j-1],
              dp[i-1][j]));
          if(dp[i][j] == dp[i][j-1]){
            backtracking[i][j] = Coor{i, j-1};
          }
          else if(dp[i][j] == dp[i-1][j-1]){
            backtracking[i][j] = Coor{i-1, j-1};
          }
          else if(dp[i][j] == dp[i-1][j]){
            backtracking[i][j] = Coor{i-1, j};
          }
          dp[i][j]++;
        }
      }
    }


    printf("%d\n", dp[strlen(s1)][strlen(s2)]);
    int curi = strlen(s1), curj = strlen(s2);
    ans.push_back(Coor{curi, curj});
    while(curi != 0 || curj != 0){
      int tempi = curi, tempj = curj;
      curi = backtracking[tempi][tempj].x; curj =
          backtracking[tempi][tempj].y;
      ans.push_back(Coor{curi, curj});
    }
    int offset = 0, cnt = 1;
    for(int i = ans.size()-2; i >= 0; i--){
      if(dp[ans[i].x][ans[i].y] != dp[ans[i+1].x][ans[i
          +1].y]){
        if((ans[i].x - ans[i+1].x) == 1 && (ans[i].y -
            ans[i+1].y) == 1){
          printf("%d Replace %d,%c\n", cnt++, ans[i].x
              + offset, s2[ans[i].y-1]);
        }
        else if((ans[i].x - ans[i+1].x) == 1 && (ans[i
            ].y - ans[i+1].y) == 0){
          printf("%d Delete %d\n", cnt++, ans[i].x+
              offset);
          offset--;
        }
        else if((ans[i].x - ans[i+1].x) == 0 && (ans[i
            ].y - ans[i+1].y) == 1){
          printf("%d Insert %d,%c\n", cnt++, ans[i].x+
              offset+1, s2[ans[i].y-1]);
          offset++;
        }
      }
    }
  }
  return 0;
}
```

String Edition

4.6 Chain Matrix Mul

```
//intut matrix的矩陣大小, output最少需做幾次乘法
int M[1005][1005];
int P[1005][1005];
vector <int> d;
int do_dp(int i, int j){
  if(M[i][j] > 0) return M[i][j];
  if(i == j) return 0;
  int minx = 0xFFFFFFFF;
  for(int k = i ; k < j ; k++){
    if((do_dp(i, k) + do_dp(k+1, j) + d[i-1]*d[k]*d[j])
        < minx){
      minx = do_dp(i, k) + do_dp(k+1, j) + d[i-1]*d[k]*
          d[j];
      P[i][j] = k;
    }
    //如果不用紀錄k是誰
    //minx = min(minx, do_dp(i, k) + do_dp(k+1, j) + d[
        i-1]*d[k]*d[j]);
  }
  return M[i][j] = minx;
}
int main(){
  int n, temp, s, ans;
  cin >> n;
  stringstream s1;
  string str;
  cin.ignore();
  while(n--){
    getline(cin, str);
    s1.clear();
    s1.str("");
    s1 << str;
    d.clear();
    while(s1 >> temp){
      d.push_back(temp);
    }
    s = d.size() - 1;
    memset(M, 0, sizeof(M));
    memset(P, 0, sizeof(P));
    ans = do_dp(1, s);
    printf("%d\n", ans);
  }
}
```

5 Search

5.1 Binary Search

```
int L = 0;     //  left boundary
int R = ans;   // right boundary
// check using L = 3, R = 4, ans = 4
while(L < R){
  int M = L + (R - L + 1) / 2;  // left + half distance
  if(ok(M)) L = M;      // ok() method is to find
      whether the M can qualify the demand
  else R = M - 1;
}

while(L < R){
  int M = L + (R - L) / 2;  // left + half distance
  if(ok(M)) R = M;      // ok() method is to find
      whether the M can qualify the demand
  else L = M + 1;
}
```

6 Sequence

6.1 RSQ(Prefix Sum)

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 10;
int arr[maxn] = {5, -2, 3, 10, -7, 1, -4, 8, -9};
int query[maxn];
void init()
{
  // every query is the sum of all previos element, include it self
  // example query[3] = arr[0] + arr[1] + arr[2] + arr[3]
  query[0] = arr[0];
  for(int i = 1; i < maxn; i++){
    query[i] = arr[i];
    query[i] += query[i-1];
  }
}
int RangeSumQuery(int s, int e)
{
  //Prefix Sum Algorithm
  if(s >= 1) return query[e] - query[s-1];
  else return query[e];
}
int main(int argc, char const *argv[])
{
  init();
  int start = 2, end = 5;
  printf("RangeSumQuery(%d, %d): %d\n", start, end, RangeSumQuery(start, end));

  return 0;
}
```

6.2 RSQ(2DPrefix Sum)

```cpp
#include <bits/stdc++.h>
using namespace std;
int arr[110][110];
int query[110][110];
int n;

int main(int argc, char const *argv[])
{
  while(cin >> n){
    // input
    for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++)
        cin >> arr[i][j];
    }
    // bulid prefix query
    for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
        query[i][j] = arr[i][j];
        if(i - 1 >= 0) query[i][j] += query[i-1][j];
        if(j - 1 >= 0) query[i][j] += query[i][j-1];
        if(i - 1 >= 0 && j - 1 >= 0) query[i][j] -= query[i-1][j-1];
      }
    }

    int temp;
    int maximum = 0x80000000;
    // find the maximum sum in any range
    for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
        for(int k = i; k < n; k++){
          for(int t = j; t < n; t++){
            temp = query[k][t];
            if(i - 1 >= 0) temp -= query[i-1][t];
            if(j - 1 >= 0) temp -= query[k][j-1];
            if(i - 1 >= 0 && j - 1 >= 0) temp += query[i-1][j-1];
            if(maximum < temp) maximum = temp;
          }
        }
      }
    }
    printf("%d\n", maximum);


  }
  return 0;
}
```

6.3 RSQ(Fenwick Tree)

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 10;
int arr[maxn] = {5, -2, 3, 10, -7, 1, -4, 8, -9};
int FenwickTree[maxn];
int ANDlowbit(int src)
{
  // src & -src will get the lowbit
  // example: 6 & -6 = 0110 & 1010 = 0010 = 2
  return src & -src;
}
void init()
{

  memset(FenwickTree, 0, sizeof(FenwickTree));
  // Notice that we start in 1
  for(int i = 1; i <= maxn; i++){
    int index = i;
    FenwickTree[i] += arr[i-1];
    int temp = arr[i-1];
    while(index + ANDlowbit(index) <= maxn){
      index += ANDlowbit(index);
      FenwickTree[index] += temp;
    }
  }
}
void Modify(int src, int val)
{
  // Modify arr[src] to val
  int gap = val - arr[src];
  arr[src] = val;
  int index = src + 1;
  FenwickTree[index] += gap;
  while(index + ANDlowbit(index) <= maxn){
    index += ANDlowbit(index);
    FenwickTree[index] += gap;
  }
}
int SequenceQuery(int src)
{
  //src is the index of the array which we want to know the Sequence Query
  int res = FenwickTree[src];
  int index = src;
  while(index - ANDlowbit(index) > 0){
    index -= ANDlowbit(index);
    res += FenwickTree[index];
  }
  return res;
}
int RangeSumQuery(int s, int e)
{
  return SequenceQuery(e) - SequenceQuery(s - 1);
}
int main(int argc, char const *argv[])
{
  init();
  int start = 2, end = 5;
  // for Fenwick index is 3, 6 for array index is 2, 5
```

```
  printf("RangeSumQuery(%d, %d): %d\n", start, end,
      RangeSumQuery(start + 1, end + 1));
  Modify(2, 5);
  // Modify arr[2] from 3 to 5
  printf("RangeSumQuery(%d, %d): %d\n", start, end,
      RangeSumQuery(start + 1, end + 1));
  return 0;
}
```

7 Sorting

7.1 Counting Sort

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 50;
const int maxDigit = 1050;
int unsorted[maxn] = {0, 3, 7, 6, 5}, sorted[maxn], aux
    [maxDigit];
// aux size is depends on the max digit in sorting
int main(int argc, char const *argv[])
{
  int n = 4;
  // array index start with 1
  memset(aux, 0, sizeof(aux));
  for(int i = 1; i <= n; i++){
    aux[unsorted[i]]++;
  }
  for(int i = 1; i < maxDigit; i++){
    aux[i] += aux[i-1];
  }
  for(int i = n; i > 0; i--){
    sorted[aux[unsorted[i]]] = unsorted[i];
    aux[unsorted[i]]--;
  }
  for(int i = 1; i <= n; i++){
    printf("%d ", sorted[i]);
  }
  return 0;
}
```

7.2 Topology Sort

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100;
vector<int> ans;
vector<int> adj[maxn];
int refs[maxn];
int n = 5;

// refs 紀錄這個點被幾個邊連到
void TopologyOrder()
{
  for(int i = 0; i < n; i++){
    int s = 0;
    while(s < n && refs[s] != 0) {
      s++;
    }
    if(s == n) break;
    refs[s] = -1;
    ans.push_back(s);
    for(auto j : adj[s]){
      refs[j]--;
    }
  }
}
int main(int argc, char const *argv[])
{
  memset(refs, 0, sizeof(refs));
  ans.clear();
  // adj[from].push_back(to); refs[to]++;
  adj[4].push_back(1); refs[1]++;
  adj[1].push_back(3); refs[3]++;
  adj[1].push_back(0); refs[0]++;
  adj[2].push_back(0); refs[0]++;
  adj[3].push_back(0); refs[0]++;
  TopologyOrder();
  for(int i = 0; i < ans.size(); i++){
    if(i == ans.size()-1) printf("%d\n", ans[i]);
    else printf("%d ", ans[i]);
  }
  return 0;
}
```

7.3 Topology Sort with DFS(check 有無環)

```
const int maxn = 5000+50;
vector<int> adj[maxn];
stack<int> ans;
int state[maxn];
bool head[maxn];
bool valid;
int n, m;

void dfs(int src)
{
    state[src] = 1;

    for (auto next : adj[src])
        if (!state[next]) dfs(next);
        else if (state[next] == 1){
            // 有環
            valid = false;
            return;
        }

    state[src] = 2;

    ans.push(src);
}

void topology_sort()
{
    for (int i = 0; i < n; i++){
        // 從 (0 ~ n-1) 找一個頭沒有被任何人連到的開始
            做dfs
        if (valid && head[i]) dfs(i);
    }

    if (!valid)
    {
        cout << "Cycle!" << endl;
        return;
    }

    while (!ans.empty())
    {
        cout << ans.top() << endl;
        ans.pop();
    }
}

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

    memset(head, true, sizeof(head));
    // make adjcent list
    for (int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;

        head[b] = false;
        //
        adj[a].push_back(b);
```

```
    }

    memset(state, 0, sizeof(state));
    valid = true;
    //如果 valid = false代表有還
    topology_sort();

    return 0;
}
```

8 Graph

8.1 DFS I

```
//implement by adjcent list
//functional dfs
void dfs(int now, int fa, int layer){
  for (auto j : adj[now])
    if(j != fa ) dfs(j, now, layer + 1);
}
//stack dfs
stack<int> st;
bool vis[maxn];
memset(vis, false, sizeof(vis));
int src;
st.push(src);
while(!st.empty())
{
  int now = st.top(); st.pop();
    vis[now] = true;
    for(auto i : adj[now])
      if(!vis[i]) st.push(i);
}
```

8.2 DFS II

```
const int maxn = 10;
struct Node{
  int d, f, color;
  // d: discover time, f: finish time, color: 0 ==
      white, 1 == gray, 2 == black
};
vector<int> adj[maxn];
Node node[maxn];
int times;
void DFS(int src)
{
  node[src].d = times++;
  node[src].color = 1;
  for(auto i : adj[src]){
    if(node[i].color == 0) DFS(i);
  }
  node[src].color = 2;
  node[src].f = times++;
}
void DFS_Start(int n, int sp)
{
  for(int i = 0; i < n; i++){
    node[i].color = 0;
  }
  times = 0;
  DFS(sp);

}
int main(int argc, char const *argv[])
{
  int n, m, x, y;
  cin >> n >> m;
  for(int i = 0; i < n; i++) adj[i].clear();
  for(int i = 0; i < m; i++){
    cin >> x >> y;
    adj[x].push_back(y);
  }
  DFS_Start(6, 0);
  for(int i = 0; i < n; i++){
    printf("%d: d: %d f: %d color: %d\n", i, node[i].d,
        node[i].f, node[i].color);
  }
  return 0;
}
```

8.3 DFS Tree

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100000+5;
vector<int> adj[maxn];
int blocks[maxn];
void dfs(int cur, int fa)
{
  blocks[cur] = 1;
  for(auto i : adj[cur]){
    if(i != fa) {
      dfs(i, cur);
      blocks[cur] += blocks[i];
    }
  }
}
int main(int argc, char const *argv[])
{
  int n, x;
  while(cin >> n){
    for(int i = 0; i <= n; i++) adj[i].clear();
    memset(blocks, 0, sizeof(blocks));
    // blocks 為包含自己，自己的子節點數量
    // 建一個無環的圖
    for(int i = 1; i < n; i++){
      cin >> x;
      adj[i].push_back(x);
      adj[x].push_back(i);
    }
    // 從0當Root做dfs
    dfs(0, -1);
    for(int i = 0; i < n; i++) {
      int ans = 0;
      for(auto j : adj[i]){
        if(blocks[i] > blocks[j]) ans = max(ans, blocks
            [j]);
      }
      ans = max(ans, n - blocks[i]);
      printf("%d\n", ans);
    }

  }
  return 0;
}
```

8.4 BFS

```
queue<int> st;
bool vis[maxn];
memset(vis, false, sizeof(vis));
int src;
st.push(src);
while(!st.empty())
{
  int now = st.front(); st.pop();
    vis[now] = true;
  for(auto i : adj[now])
      if(!vis[i]) st.push(i);
}
```

8.5 AOE

```cpp
struct AOE {
    // zero base
    const int INF = 1e9;
    struct Edge {
        int at;
        int cost;
    };

    struct Vertex {
        int early;
        int late;
        vector<Edge> from;
        vector<Edge> to;
    };

    int n;
    vector<Vertex> vertices;

    void init(int _n) {
        n = _n;
        vertices.clear();
        vertices.resize(_n);
        for (int i = 0; i < n; i++) {
            vertices[i].early = -1;
            vertices[i].late = INF;
        }
    }

    void addEdge(int from, int to, int cost) {
        // zero base
        vertices[from].to.push_back({to, cost});
        vertices[to].from.push_back({from, cost});
    }

    void dfsEarly(int now) {
        for (auto e : vertices[now].to) {
            if (vertices[e.at].early < vertices[now].
                early + e.cost) {
                vertices[e.at].early = vertices[now].
                    early + e.cost;
                dfsEarly(e.at);
            }
        }
    }

    void dfsLate(int now) {
        for (auto e : vertices[now].from) {
            if (vertices[e.at].late > vertices[now].
                late - e.cost) {
                vertices[e.at].late = vertices[now].
                    late - e.cost;
                dfsLate(e.at);
            }
        }
    }

    // may be slow?
    void printCritical(int now, vector<int> path) {
        if (vertices[now].to.size() == 0) {
            // critical path found
            for (auto i : path) {
                cout << i << ' ';
            }
            cout << '\n';
            return;
        }
        for (auto e : vertices[now].to) {
            if (vertices[e.at].early == vertices[e.at].
                late) {
                vector<int> tmp = path;
                tmp.push_back(e.at);
                printCritical(e.at, tmp);
            }
        }
    }

    int run() {
        for (int i = 0; i < n; i++) {
            if (vertices[i].from.size() == 0) {
                vertices[i].early = 0;
                dfsEarly(i);
            }
        }

        int ans = 0;
        for (int i = 0; i < n; i++) {
            if (vertices[i].to.size() == 0) {
                vertices[i].late = vertices[i].early;
                ans = max(ans, vertices[i].late);
                dfsLate(i);
            }
        }

        for (int i = 0; i < n; i++) {
            cout << "i = " << i << " early = " <<
                vertices[i].early << " late = " <<
                vertices[i].late << "\n";
        }

        for (int i = 0; i < n; i++) {
            if (vertices[i].from.size() == 0) {
                vector<int> path;
                path.push_back(i);
                printCritical(i, path);
            }
        }

        return ans;
    }
};
int main() {
    AOE aoetest;
    int n, m;
    cin >> n >> m;
    aoetest.init(n);

    int a, b, w;
    for (int i = 0; i < m; i++) {
        cin >> a >> b >> w;
        aoetest.addEdge(a, b, w);
    }

    int res = aoetest.run();
    cout << "res = " << res << endl;
}
```

8.6 Dijkstra

```cpp
#define MP make_pair
#define PII pair<int, int>
#define maxn 50000 + 5

int dis[maxn];     // 預設都是 INF
vector<PII> adj[maxn]; // (連到的點， 邊的距離)

void dijk(int cur) // dijk(起點)
{
  int d;
  priority_queue<PII, vector<PII>, greater<PII>> q; //
      放 (距離, 點編號)，每次會拿距離最小的點出來
  q.push(MP(0, cur));

  while (!q.empty())
  {
    tie(d, cur) = q.top(); q.pop();
    if (dis[cur] != 1e9) continue; // 如果之前就拜訪
        過，無視

    dis[cur] = d;

    for (auto i : adj[cur]){
```

```
      if (dis[i.first] == 1e9) q.push(MP(d + i.second, i.first));
    }

  }
}

void init(void)
{
  fill(dis, dis + maxn, 1e9);

  for (int i = 0; i < maxn; i++){
    adj[i].clear();
  }
}
```

8.7 SPFA

```
#include <bits/stdc++.h>
using namespace std;

#define INF 0x3f3f3f3f
const int maxn = 10000+5;

int n, m;
int dist[maxn], out[maxn];
bool inq[maxn];
//dist = distance, inq = inqit, out
vector<pair<int, int>> adj[maxn];

void init()
{
  memset(dist, INF, sizeof(dist));
  memset(inq, 0, sizeof(inq));
  memset(out, 0, sizeof(out));
  for(int i = 0; i <= n;  i++){
    adj[i].clear();
  }
}

bool spfa(int sp, int n)
{
  queue<int> q;
  q.push(sp);
  dist[sp] = 0; inq[sp] = true;

  while(!q.empty())
  {
    int u = q.front(); q.pop();
    inq[u] = false; // pop point
    out[u]++;
    if(out[u] > n) return false; // negative cycle occurs

    for(int j = 0; j < adj[u].size(); j++){
      int v = adj[u][j].first;  // first is point, second is weight
      if(dist[v] > dist[u] + adj[u][j].second){
        dist[v] = dist[u] + adj[u][j].second;
        if(inq[v]) continue;

        inq[v] = true;  //push point
        q.push(v);
      }
    }
  }
  return true;
}

int main(int argc, char const *argv[])
{
  // n nodes and m edges
  scanf("%d%d", &n, &m);
  init();
  // make adjcent list
  int a, b, w;
  for(int i = 0; i < m; i++){
    scanf("%d%d%d", &a, &b, &w);
    adj[a].push_back(make_pair(b, w));
  }
  int sp = 0; // start point

  if(spfa(sp, n))
    for (int i = 0; i < n; i++) printf("dist %d: %d\n", i, dist[i]);
  else printf("can't reach.\n");
  return 0;
}
```

8.8 BellmanFord

```
int main(int argc, char const *argv[])
{
  //initialize dis[] with 1e9
  //make an adjecnt list
  call bellman_ford(src);
  return 0;
}

void bellman_ford(int src)
{
  dis[src] = 0;                  //initialize source with distance 0
  for (int k = 0; k < n - 1; ++k){        //do n-1 times
    for (int i = 0; i < n; ++i){
      for(auto j : v[i]){
        if(dis[i] != 1e9) dis[j] = min(dis[j], dis[i] + w[i][j]);
      }
    }
   }
}
bool negativeCycle()
{
  for(i = 0; i < n; ++i){
    for(auto j : v[i]){
      if(dis[j] > dis[i] + w[i][j]) return true //has negative cycle
    }
  }
  return false;
}
```

8.9 FloydWarshall

```
//dis[i][j] is the distance of node i to node j
int dis[n+5][n+5];
void init()
{
  memset(dis, 0x3f, sizeof(dis));
  for(int i = 0; i < n; i++) d[i][i] = 0;
}
void floyd(){
  for (int k = 0; k < n; ++k)
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < n; ++j)
        dis[i][j] = dis[j][i] = min(dis[i][j],  dis[i][k] + dis[k][j]);
}
int main(int argc, char const *argv[])
{
  //If we got n nodes, label from 0 to (n-1)
  init();
  //Set the dis
  floyd();
}
```

8.10 Kruskal

```cpp
const int maxn = 1000+5;
struct Edge
{
  int from, to;
  double cost;
  bool operator<(const Edge other){
    return cost < other.cost;
  }
}E[maxn*maxn];
int p[maxn];
vector<Edge> G[maxn];
int find(int x){
    return p[x] < 0 ? x : (p[x] = find(p[x]));
}
bool uni(int x ,int y)
{
  int xRoot = find(x), yRoot = find(y);
  if(xRoot != yRoot){
    if(p[xRoot] < p[yRoot]){
      p[xRoot] += p[yRoot];
      p[yRoot] = xRoot;
    }
    else{
      p[yRoot] += p[xRoot];
      p[xRoot] = yRoot;
    }
    return true;
  }
  else return false;
}
double kruskal(int n, int m)
{
  // n is the numbers of node, m is the numbers of edge.
  for(int i = 0; i <= n; i++){
    G[i].clear();
    p[i] = -1;
  }
  sort(E, E + m);
  double ans = 0;
  int edge_cnt = 0;
  for(int i = 0; i < m; i++){
    if(uni(E[i].from, E[i].to)){
      int from = E[i].from, to = E[i].to;
      ans += E[i].cost;
      G[from].push_back(Edge{from, to, E[i].cost});
      G[to].push_back(Edge{to, from, E[i].cost});
      if(++edge_cnt == n-1) break;
    }
  }
  if(edge_cnt == n-1) return ans;
  else return -1;// means can't found spanning tree
}
// find max segment in MST graph
int maxcost[maxn][maxn];
vector<int> visited;
void dfs(int pre, int now, int w){
  for(auto x : visited){
    maxcost[x][now] = maxcost[now][x] = max(w, maxcost[pre][x]);
  }
  visited.push_back(now);
  for(auto i : G[now]){
    if(pre != i.to) dfs(now, i.to, i.cost);
  }
}
void findMaxPtah(int sp, int ep){
  memset(maxcost, 0,  sizeof(maxcost));
  visited.clear();
  dfs(-1, sp, 0);
}
```

8.11 Articulation Point

```cpp
#define NIL -1
// A class that represents an undirected graph
class Graph
{
    int V;          // No. of vertices
    list<int> *adj; // A dynamic array of adjacency lists
    void APUtil(int v, bool visited[], int disc[], int low[],
                int parent[], bool ap[]);

public:
    Graph(int V);               // Constructor
    void addEdge(int v, int w); // function to add an edge to graph
    void AP();                  // prints articulation points
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
    adj[w].push_back(v); // Note: the graph is undirected
}

// A recursive function that find articulation points using DFS traversal
// u --> The vertex to be visited next
// visited[] --> keeps tract of visited vertices
// disc[] --> Stores discovery times of visited vertices
// parent[] --> Stores parent vertices in DFS tree
// ap[] --> Store articulation points
void Graph::APUtil(int u, bool visited[], int disc[],
                   int low[], int parent[], bool ap[])
{
    // A static variable is used for simplicity, we can avoid use of static
    // variable by passing a pointer.
    static int time = 0;

    // Count of children in DFS Tree
    int children = 0;

    // Mark the current node as visited
    visited[u] = true;

    // Initialize discovery time and low value
    disc[u] = low[u] = ++time;

    // Go through all vertices aadjacent to this
    list<int>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i)
    {
        int v = *i; // v is current adjacent of u

        // If v is not visited yet, then make it a child of u
        // in DFS tree and recur for it
        if (!visited[v])
        {
            children++;
            parent[v] = u;
            APUtil(v, visited, disc, low, parent, ap);

            // Check if the subtree rooted with v has a connection to
            // one of the ancestors of u
            low[u] = min(low[u], low[v]);

```

```
            // u is an articulation point in following cases

            // (1) u is root of DFS tree and has two or more chilren.
            if (parent[u] == NIL && children > 1)
                ap[u] = true;

            // (2) If u is not root and low value of one of its child is more
            // than discovery value of u.
            if (parent[u] != NIL && low[v] >= disc[u])
                ap[u] = true;
        }

        // Update low value of u for parent function calls.
        else if (v != parent[u])
            low[u] = min(low[u], disc[v]);
    }
}

// The function to do DFS traversal. It uses recursive function APUtil()
void Graph::AP()
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    int *disc = new int[V];
    int *low = new int[V];
    int *parent = new int[V];
    bool *ap = new bool[V]; // To store articulation points

    // Initialize parent and visited, and ap(articulation point) arrays
    for (int i = 0; i < V; i++)
    {
        parent[i] = NIL;
        visited[i] = false;
        ap[i] = false;
    }

    // Call the recursive helper function to find articulation points
    // in DFS tree rooted with vertex 'i'
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            APUtil(i, visited, disc, low, parent, ap);

    // Now ap[] contains articulation points, print them
    for (int i = 0; i < V; i++)
        if (ap[i] == true)
            cout << i << " ";
}

int main()
{
    Graph g(7);
    g.addEdge(0, 1);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(1, 6);
    g.addEdge(3, 5);
    g.addEdge(4, 5);
    g.AP();

    return 0;
}
```

8.12 KM

```
template <typename T>
struct KM
{
    int n;
    int Left[N];
    T w[N][N], Lx[N], Ly[N];
    bitset<N> vx, vy;

    void init(int _n)
    {
        n = _n;
    }

    bool match(int i)
    {
        vx[i] = true;
        for (int j = 1; j <= n; j++)
        {
            if ((fabs(Lx[i] + Ly[j] - w[i][j]) < 1e-9)
                && !vy[j])
            {
                vy[j] = 1;
                if (!Left[j] || match(Left[j]))
                {
                    Left[j] = i;
                    return true;
                }
            }
        }
        return false;
    }

    void update()
    {
        T a = 1e9;
        for (int i = 1; i <= n; i++)
        {
            if (vx[i])
            {
                for (int j = 1; j <= n; j++)
                {
                    if (!vy[j])
                    {
                        a = min(a, Lx[i] + Ly[j] - w[i][j]);
                    }
                }
            }
        }
        for (int i = 1; i <= n; i++)
        {
            if (vx[i])
            {
                Lx[i] -= a;
            }
            if (vy[i])
            {
                Ly[i] += a;
            }
        }
    }

    void hungarian()
    {
        for (int i = 1; i <= n; i++)
        {
            Left[i] = Lx[i] = Ly[i] = 0;
            for (int j = 1; j <= n; j++)
            {
                Lx[i] = max(Lx[i], w[i][j]);
            }
        }
        for (int i = 1; i <= n; i++)
        {
            while (1)
            {
                vx.reset();
                vy.reset();
```

```
                if (match(i))
                {
                    break;
                }
                update();
            }
        }
    }
};

/*
usage
KM<int> km; // declare with weight type
km.init(n); // initialize with vertex
km.hungarian(); // calculate
km.w[][]; // weight array
km.Left[i] // y_i match x_Left[i]
*/
```

8.13 Bipartite Matching

```
const int maxn = 500+5;
int W[maxn][maxn], n;
int Lx[maxn], Ly[maxn];
int Lef[maxn];
bool S[maxn], T[maxn];
bool match(int i)
{
  S[i] = true;
  for (int j = 1; j <= n; ++j)
  {
    if(Lx[i] + Ly[j] == W[i][j] && !T[j])
    {
      T[j] = true;
      if(!Lef[j] || match(Lef[j]))
      {
        Lef[j] = i;

        return true;
      }
    }
  }
  return false;
}
void update()
{
  int a = 0x3f3f3f3f;
  for(int i = 1; i <= n; i++)
  {
    if(S[i])
    {
      for(int j = 1; j <= n; j++)
      {
        if(!T[j]) a = min(a, Lx[i] + Ly[j] - W[i][j]);
      }
    }
  }
  for(int i = 1; i <= n; i++)
  {
    if(S[i]) Lx[i] -= a;
    if(T[i]) Ly[i] += a;
  }
}
void KM()
{
  for (int i = 1; i <= n; ++i)
  {
    Lef[i] = Lx[i] = Ly[i] = 0;
    for(int j = 1; j <= n; j++){
      Lx[i] = max(Lx[i], W[i][j]);
    }
  }
  for (int i = 1; i <= n; ++i)
  {
    for(;;){
      for(int j = 1; j <= n; j++){
        S[j] = T[j] = 0;
      }
      if(match(i)) break;
      else update();
    }


  }
}
int main(int argc, char const *argv[])
{
  for(int i = 1; i <= n; i++){
    for(int j = 1; j <= n; j++){
      scanf("%d", &W[i][j]);
    }
  }

  KM();
  int ans = 0;

  for(int i = 1; i <= n; i++){
    ans += Ly[i];
    ans += Lx[i];
  }

  for(int i = 1; i <= n; i++){
    if(i != n) printf("%d ", Lx[i]);
    else printf("%d\n", Lx[i]);
  }

  for(int i = 1; i <= n; i++){
    if(i != n) printf("%d ", Ly[i]);
    else printf("%d\n", Ly[i]);
  }

  printf("%d\n", ans);
  return 0;
}
```

8.14 Bipartite Checkl

```
#define S 50050

vector<int> map[S];
int visit[S];
bool valid;

void check(int start)
{
    stack<int> st;
    st.push(start);
    visit[start] = 1;

    while (valid && !st.empty())
    {
        int cur = st.top();
        st.pop();

        for (int i = 0; i < map[cur].size(); i++)
        {
            int next = map[cur][i];

            if (visit[next] == -1)
            {
                st.push(next);

                if (visit[cur] == 1)
                    visit[next] = 2;
                else
                    visit[next] = 1;
            }
            else if (visit[cur] == visit[next])
                valid = false;
        }
    }
}
```

```

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

    for (int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;

        map[a].push_back(b);
        map[b].push_back(a);
    }

    // -1 : not visit, 1 : tsudere, 2 : proud
    memset(visit, -1, sizeof(visit));
    valid = true;

    for (int i = 1; i <= n; i++)
    {
        if (valid && visit[i] == -1)
        {
            check(i);
        }
    }

    if (valid)
        cout << "yes" << endl;
    else
        cout << "no" << endl;

    return 0;
}
```

8.15 CLE Directed MST

```
const int maxn = 60+5;
const int INF = 0x3f3f3f3f;
struct Edge
{
  int from, to, cost;
};
Edge E[maxn * maxn], e[maxn * maxn];
int n, m, c;
int in[maxn], pre[maxn], id[maxn], vis[maxn];
int CLE(int root, int n, int m)
{
  int res = 0;
  while(1)
  {
    for(int i = 0; i < n; i++){
      in[i] = INF;
    }
    //Find in edge
    for(int i = 0; i < m; i++){
      int from = e[i].from, to = e[i].to;
      if(from != to && e[i].cost < in[to]){
        in[to] = e[i].cost;
        pre[to] = from;
      }
    }
    //Check in edge
    for(int i = 0; i < n; i++){
      if(i == root) continue;
      if(in[i] == INF) return -1;
    }

    int num = 0;
    memset(id, -1, sizeof(id));
    memset(vis, -1, sizeof(vis));
    in[root] = 0;

    //Find cycles
    for(int i = 0; i < n; i++){
      res += in[i];
      int v = i;
      while(vis[v] != i && id[v] == -1 && v != root)
      {
        vis[v] = i;
        v = pre[v];
      }
      if(v != root && id[v] == -1)
      {
        for(int j = pre[v]; j != v; j = pre[j]){
          id[j] = num;
        }
        id[v] = num++;
      }
    }
    //No cycle
    if(num == 0) break;
    for(int i = 0; i < n; i++){
      if(id[i] == -1) id[i] = num++;
    }
    //Grouping the vertices
    for(int i = 0; i < m; i++){
      int from = e[i].from, to = e[i].to;
      e[i].from = id[from]; e[i].to = id[to];
      if(id[from] != id[to]) e[i].cost -= in[to];
    }
    n = num;
    root = id[root];
  }
  return res;
}
int main(int argc, char const *argv[])
{
  int n, m;
  // n nodes and m edges
  scanf("%d%d", &n, &m);
  for(int i = 0; i < m; i++){
    scanf("%d%d%d%d", &E[i].from, &E[i].to, &E[i].cost)
        ;
  }
  int sp = 0; // start point
  int ans = CLE(sp, n, m);
  if(ans == -1) printf("No Directed Minimum Spanning
      Tree.\n");
  else printf("%d\n", ans);
  return 0;
}
```

8.16 Dinic

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 50+5;
const int INF = 0x3f3f3f3f;
template <typename T>
struct Dinic
{
    int n, s, t, level[maxn], now[maxn];
    struct Edge
    {
        int v;
        T rf; // rf: residual flow
        int re;
    };
    vector<Edge> e[maxn];
    void init(int _n, int _s, int _t)
    {
        n = _n;
        s = _s;
        t = _t;
        for (int i = 0; i <= n; i++)
        {
            e[i].clear();
        }
    }
    void add_edge(int u, int v, T f)
    {
```

```
        e[u].push_back({v, f, (int)e[v].size()});
        e[v].push_back({u, f, (int)e[u].size() - 1});
        // for directional graph
        // e[v].push_back({u, 0, (int)e[u].size() - 1})
            ;
    }
    bool bfs()
    {
        fill(level, level + n + 1, -1);
        queue<int> q;
        q.push(s);
        level[s] = 0;
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (auto it : e[u])
            {
                if (it.rf > 0 && level[it.v] == -1)
                {
                    level[it.v] = level[u] + 1;
                    q.push(it.v);
                }
            }
        }
        return level[t] != -1;
    }
    T dfs(int u, T limit)
    {
        if (u == t)
            return limit;
        T res = 0;
        while (now[u] < (int)e[u].size())
        {
            Edge &it = e[u][now[u]];
            if (it.rf > 0 && level[it.v] == level[u] +
                1)
            {
                T f = dfs(it.v, min(limit, it.rf));
                res += f;
                limit -= f;
                it.rf -= f;
                e[it.v][it.re].rf += f;
                if (limit == 0)
                {
                    return res;
                }
            }
            else
            {
                ++now[u];
            }
        }
        if (!res)
        {
            level[u] = -1;
        }
        return res;
    }
    T flow(T res = 0)
    {
        while (bfs())
        {
            T tmp;
            memset(now, 0, sizeof(now));
            do{
                tmp = dfs(s, INF);
                res += tmp;
            }while(tmp);
        }
        return res;
    }
};

/*
usage
Dinic<int> dinic; // declare, flow type is int
dinic.init(n, s, t); // initialize, n vertexs, start
    from s to t
dinic.add_edge(x, y, z); // add edge from x to y,
    weight is z
dinic.flow() // calculate max flow
*/
```

8.17 MCMF

```
struct Edge
{
    int v;
    T cost;
    int cap;
    Edge(int _v, int _cost, int _cap) : v(_v), cost(
        _cost), cap(_cap) {}
};
vector<int> dis(MXV), pre(MXV);
vector<vector<int>> G(MXV);
int n;
vector<Edge> edges;
bitset<MXV> inque;
queue<int> q;
void init(int _n)
{
    n = _n;
    edges.clear();
    for (int i = 0; i <= MXV; ++i)
    {
        G[i].clear();
    }
}
void addEdge(int u, int v, T cost, int cap)
{
    G[u].push_back((int)edges.size());
    edges.push_back(Edge(v, cost, cap));
    G[v].push_back((int)edges.size());
    edges.push_back(Edge(u, -cost, 0));
}
bool spfa(int s, int t)
{
    FOR(i, 0, MXV) { dis[i] = INF; }
    inque.reset();
    while (!q.empty())
    {
        q.pop();
    }
    dis[s] = 0;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        inque[u] = false;
        FOR(i, 0, G[u].size())
        {
            Edge &e = edges[G[u][i]];
            if (e.cap > 0 && dis[e.v] > dis[u] + e.cost
                )
            {
                dis[e.v] = dis[u] + e.cost;
                pre[e.v] = G[u][i];
                if (!inque[e.v])
                {
                    q.push(e.v);
                    inque[e.v] = true;
                }
            }
        }
    }
    return dis[t] != INF;
}
void update(int s, int t, int bottleneck)
{
    for (int u = t; u != s;)
    {
```

```
66          int pos = pre[u];
67          edges[pos].cap -= bottleneck;
68          edges[pos ^ 1].cap += bottleneck;
69          u = edges[pos ^ 1].v;
70      }
71  }
72  void sol(int s, int t)
73  {
74      int mnCost = 0;
75      while (spfa(s, t))
76      {
77          update(s, t, 1);
78          mnCost += dis[t];
79      }
80      cout << mnCost << '\n';
81  }
82
83  int main()
84  {
85      IOS;
86      int n, m;
87      while (cin >> n >> m)
88      {
89          init(n);
90          for (int i = 0, f, t, w; i != m; ++i)
91          {
92              cin >> f >> t >> w;
93              addEdge(f, t, w, 1);
94              addEdge(t, f, w, 1);
95          }
96          int s = 0, t = n + 1;
97          addEdge(s, 1, 0, 2);
98          addEdge(n, t, 0, 2);
99          sol(s, t);
100     }
101 }
```

9 Number

9.1 Sieve

```
1  const int maxn = 500+10;
2  bool visit[maxn];
3  int primes[maxn];
4  int sieve(int src)
5  {
6    memset(visit, false, sizeof(visit));
7    for(int i = 2; i <= sqrt(src + 0.5); i++){
8      if(!visit[i]){
9        for(int j = i * i; j <= src; j += i){
10         visit[j] = true;
11       }
12     }
13   }
14   int cnt = 0;
15   for(int i = 2; i <= src; i++){
16     if(!visit[i]) primes[cnt++] = i;
17   }
18   return cnt;
19 }
```

9.2 Power

```
1  double Power(double x, int n)
2  {
3      if (n == 0) return 1.00;
4      if (n == 1) return x;
5      double ans = Power(x, n / 2);
6      if (n % 2 == 0) return ans * ans;
7      else if (n < 0) return ans * ans / x;
8      else return ans * ans * x;
9  }
```

9.3 Euler

```
1  const int maxn = 50000;
2  int F[maxn+5];
3  void Euler(){
4    memset(F, 0, sizeof(F));
5    F[1] = 1;
6    for(int i=2; i<maxn; i++){
7      if(!F[i]){
8        for(int j=i; j<maxn; j+=i){
9          if(!F[j]) F[j] = j;
10         F[j] = F[j] / i*(i-1);
11       }
12     }
13   }
14 }
```

9.4 Factors

```
1  vector<int> getDivisiors(int x){
2    vector<int> res;
3    int sq = (int) sqrt(x + 0.5);
4    for(int i = 1; i <= sq; i++){
5      if(x % i == 0) {
6        int j = x / i;
7        res.push_back(i);
8        if(i != j) res.push_back(j);
9      }
10   }
11   return res;
12 }
```

9.5 Extend Euclidean

```
1  int extgcd(int a, int b, int &x, int &y)
2  {
3      int d = a;
4      if (b)
5      {
6          d = extgcd(b, a % b, y, x), y -= (a / b) * x;
7      }
8      else
9          x = 1, y = 0;
10     return d;
11 } // ax+by=1 ax同餘 1 mod b
```

9.6 Matrix

```
1  const int maxn = 20;
2  struct Matrix  {
3      long long a[maxn][maxn];
4      void zeroM() // 清空矩陣
5      {
6          memset(a, 0, sizeof(a));
7      }
8      void identityM()
9      {
10         memset(a, 0, sizeof a);
11         for(int i = 0; i < maxn; i++)
12             a[i][i] = 1;
13     }
14 };
15 int n; // n * n matrix
16
17 Matrix operator*(const Matrix &a, const Matrix &b) //
       矩陣乘法
18 {
19     Matrix ret;
20     ret.all_0();
21     for (int i = 0; i < n; i++){
```

```
        for (int j = 0; j < n; j++){
            for (int k = 0; k < n; k++){
                ret.a[i][j] += a.a[i][k] * b.a[k][j];
                ret.a[i][j] %= m;
            }
        }
    }
    return ret;
}

Matrix mpower(Matrix a, int n)
{
    Matrix ret;
    ret.identityM();
    if (n==0)  return ret;
    if (n==1)  return a;
    ret = mpower(a, n/2);
    ret = ret * ret;
    if (n % 2 == 1)  ret = ret * a;
    return ret;
}
int main(int argc, char const *argv[])
{
    cin >> n; // n * n matrix
    Matrix A; // declare
    return 0;
}
```

9.7 GaussElimination

```
const int MAXN = 300;
const double EPS = 1e-8;
int n;
double A[MAXN][MAXN];
void Gauss()
{
    for (int i = 0; i < n; i++)
    {
        bool ok = 0;
        for (int j = i; j < n; j++)
        {
            if (fabs(A[j][i]) > EPS)
            {
                swap(A[j], A[i]);
                ok = 1;
                break;
            }
        }
        if (!ok)
            continue;
        double fs = A[i][i];
        for (int j = i + 1; j < n; j++)
        {
            double r = A[j][i] / fs;
            for (int k = i; k < n; k++)
            {
                A[j][k] -= A[i][k] * r;
            }
        }
    }
}
```

10 Geometry

10.1 Geometry

```
using f64 = double;
using dvt = f64;
const double eps = 1e-9;
struct dot
{
  dvt x, y;
};

struct line
{
  dot start, end;
};

dot operator+(dot a, dot b) { return {a.x + b.x, a.y + b.y}; }
dot operator-(dot a, dot b) { return {a.x - b.x, a.y - b.y}; }
dot operator*(dot a, dvt c) { return {a.x * c, a.y * c}; }
dot operator*(dvt c, dot a) { return a * c; }
dot operator/(dot a, dvt c) { return {a.x / c, a.y / c}; }
bool operator<(dot a, dot b)
{
  return std::tie(a.x, a.y) < std::tie(b.x, b.y);
}

dvt iproduct(dot a, dot b)
{
  return a.x * b.x + a.y * b.y;
}

dvt cross(dot a, dot b)
{
  return a.x * b.y - a.y * b.x;
}

int side(line L, dot a)
{
  dvt cross_value = cross(a - L.start, L.end - L.start);
  if (cross_value > eps)
  {
    return 1;
  }
  else if (cross_value < -eps)
  {
    return -1;
  }
  return 0;
}

bool has_jiao(line AB, line CD)
{
  int c = side(AB, CD.start);
  int d = side(AB, CD.end);
  // 0 代表在線上
  if (c == 0 || d == 0)
    return true;
  // 正負號不同=>異側=>相交
  return c == -d;
}
```

10.2 Lines Intersection1

```
#include <iostream>
#include <cmath>
#include <cstring>

using namespace std;

struct pt {
  double x, y;
};

struct line {
  double a, b, c;
  line(pt p1, pt p2) {
    a = p2.y - p1.y;
    b = p1.x - p2.x;
    c = -a * p1.x - b * p1.y;
```

```
  }
};

const double EPS = 1e-9;

double det (double a, double b, double c, double d) {
  return a * d - b * c;
}

bool intersect (line m, line n, pt & res) {
  double zn = det (m.a, m.b, n.a, n.b);
  if (abs (zn) < EPS)
    return false;
  res.x = - det (m.c, m.b, n.c, n.b) / zn;
  res.y = - det (m.a, m.c, n.a, n.c) / zn;
  return true;
}

bool parallel (line m, line n) {
  return abs (det (m.a, m.b, n.a, n.b)) < EPS;
}

bool equivalent (line m, line n) {
  return abs (det (m.a, m.b, n.a, n.b)) < EPS
    && abs (det (m.a, m.c, n.a, n.c)) < EPS
    && abs (det (m.b, m.c, n.b, n.c)) < EPS;
}

void solve(line a, line b) {
  if (equivalent(a, b)) {
    cout << "LINE\n";
    return ;
  }
  if (parallel(a, b)) {
    cout << "NONE\n";
    return ;
  }
  pt res;
  intersect(a, b, res);
  cout.precision(2);
  cout << "POINT " << fixed << res.x << " " << res.y << "\n";
}

int main() {
  int t;
  cin >> t;
  cout << "INTERSECTING LINES OUTPUT\n";
  while (t--) {
      pt p1, p2;
    cin >> p1.x >> p1.y >> p2.x >> p2.y;
    line a = line(p1, p2);
    cin >> p1.x >> p1.y >> p2.x >> p2.y;
    line b = line(p1, p2);

    solve(a, b);
  }
  cout << "END OF OUTPUT\n";
    return 0;
}
```

10.3 Lines Intersection2

```
#include<iostream>
#include<cstdio>
using namespace std;

struct point {
    int x, y;
};

int direction (point p1, point p2, point p3) {
    return (p2.y - p1.y) * (p3.x - p2.x) - (p3.y - p2.y) * (p2.x - p1.x);
}

bool onSeg (point P1, point P2, point P3) {
    if ( min(P1.x, P2.x) <= P3.x && P3.x <= max(P1.x, P2.x) )
        if ( min(P1.y, P2.y) <= P3.y && P3.y <= max(P1.y, P2.y) )
            return true;
    return false;
}

bool segIntersect (point p1, point p2, point p3, point p4) {

    int d1 = direction(p3,p4,p1); int d2 = direction(p3,p4,p2);
    int d3 = direction(p1,p2,p3); int d4 = direction(p1,p2,p4);

    if ( ((d1>0 && d2<0) || (d1<0 && d2>0)) && ((d3>0 && d4<0) || (d3<0 && d4>0)) )
        return true;
    else if ( d1 == 0 && onSeg(p3,p4,p1) ) return true;
    else if ( d2 == 0 && onSeg(p3,p4,p2) ) return true;
    else if ( d3 == 0 && onSeg(p1,p2,p3) ) return true;
    else if ( d4 == 0 && onSeg(p1,p2,p4) ) return true;
    else return false;

}

int main () {

    // We will have four point to check the lines
    // 2 for one lines edges, 2 for the other one
    point line1p1;
    point line1p2;
    point line2p1;
    point line2p2;

    // Inputs for points
    cin >> line1p1.x >> line1p1.y >> line1p2.x >> line1p2.y;
    cin >> line2p1.x >> line2p1.y >> line2p2.x >> line2p2.y;

    if ( segIntersect( line1p1, line1p2, line2p1, line2p2 ) )
        cout << "YES" << endl ;
    else
        cout << "NO" << endl ;

}
```

10.4 Polygon Inside Or Outside

```
#include<iostream>
#define INF 10000
using namespace std;

struct point {
  int x,y;
};

int dir (point p1, point p2, point p3) {

  int val = (p3.x - p2.x) * (p2.y - p1.y) - (p3.y - p2.y) * (p2.x - p1.x);

  if (val == 0) return 0;
  return (val > 0) ? 1 : 2;

}

bool onSeg (point p1, point p2, point p3) {

  if ((min(p1.x , p2.x) <= p3.x && p3.x <= max(p1.x, p2.x)) && (min(p1.y, p2.y) <= p3.y && p3.y <= max(p1.y, p2.y)) ) return true;
```

```
  return false;

}

bool doIntersect (point p1, point p2, point p3, point p4) {

  int d1 = dir(p3, p4, p1); int d2 = dir(p3, p4, p2);
  int d3 = dir(p1, p2, p3); int d4 = dir(p1, p2, p4);

  if (d1 != d2 && d3 != d4) return true;
  if (d1 == 0 && onSeg(p3, p4, p1)) return true;
  if (d2 == 0 && onSeg(p3, p4, p2)) return true;
  if (d3 == 0 && onSeg(p1, p2, p3)) return true;
  if (d4 == 0 && onSeg(p1, p2, p4)) return true;
  return false;

}

bool inInside ( point * polygon, int n, point p) {

  if ( n < 3 ) return false;

  point extreme = { INF , p.y };

  int count = 0, i = 0;

  do{

    int next = (i+1)%n;

    if(doIntersect(polygon[i], polygon[next], p, extreme)) {
      if(dir(polygon[i], p, polygon[next]) == 0)
          return onSeg(polygon[i], p, polygon[next]);
              // Might be wrong I will check
      count++;
    }

    i = next;
  } while(i != 0);

  return count&1;

}

int main () {

    point Polygon[4] = {{0,0},{0,4},{4,4},{4,0}};
    int size = sizeof(Polygon)/sizeof(point);

    point p = {20, 20};
    inInside(Polygon, size, p) ? cout << "Yes\n" : cout << "No\n" ;

    point p2 = {2, 2};
    inInside(Polygon, size, p2) ? cout << "Yes\n" : cout << "No\n" ;


}
```

```
}//若點的angle一樣，則比較遠的點

bool find_small_vertex(point a, point b) {
  return (a.y < b.y) || (a.y == b.y && a.x < b.x);
}

int cross(point o, point a, point b) {
  return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}

bool compare_angle(point a, point b){
  double c = cross( p[0], a, b );
  if ( !c ) return a.d < b.d;
  else return c > 0;
}

void GrahamScan(int k){
  sort(p+0, p+k, find_small_vertex);
  for(int i=1; i<k; i++){
    p[i].d = dist(p[0], p[i]);
  }
  sort(p+1, p+k, compare_angle);

  int m=0;
  for(int i=0; i<k; i++){
    while(m>=2 && cross(ch[m-2], ch[m-1], p[i]) <= 0){
      m--;
    }
    ch[m++] = p[i];
  }
  // Convex Hull find m nodes and print them out
  printf("%d\n", m+1);
  for(int j=0; j<m; j++){
    printf("%d %d\n", ch[j].x, ch[j].y);
  }
  printf("%d %d\n", ch[0].x, ch[0].y);
}
```

10.5 Convex Hull

```
#include <bits/stdc++.h>
using namespace std;

struct point{
  int x;
  int y;
  int d;
}p[600],ch[600];

int dist(point a, point b) {
  return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
```