1 Recursion

1.1 DFS

```
Function dfs(G,u)
  color(u) = gray
  for each (u,v) ∈ G and color(v) = white do
    dfs(G,v)
  color(u) = black
```

```
void dfs(int node){
    strcpy(graph[node-1].color, "gray");
    aux++;
    graph[node-1].dfs = aux;
    int i;
    for(i = 0; i < graph[node-1].n_edges; i++){
        if(strcmp(graph[graph[node-1].edges[i]-1].color, "white") == 0){
            dfs(graph[node-1].edges[i]);
            aux--;
        }
    }
    strcpy(graph[node-1].color, "black");
}
```

2 Backtracking

2.1 Template

```
Function BT(s)
  if reject(s) = true then
    return false
  if accept(s) = true then
    output(s)
    return true
  while condition(s) = true do
    s‘ = update(s)
    if BT(s‘) = true then
      return true
  return false
```

2.2 8 Queens

```
Function nQueens(col)
  if col = N + 1 then
    return true
  for i = 1 to N do
```

```
    Q[col] = i
    if attack(col) = false then
      if nQueens(col + 1) = true then
        return true
  return false
```

2.3 Hamiltonian path

```
Function HamPath(v)
  if v = t then
    if sum(visit) = N then
      return true
    else
      return false
  for (v, i) ∈ G do
    if visit[i] = 0 then
      visit[i] = 1
      if HamPath(i) = true then
        return true
      visit[i] = 0
    return false
```

2.4 Hamiltonian cycle

```
Function HamCycle(v)
  if sum(visit) = N then
    if (v, 1) ∈ G then
      return true
  for (v, i) ∈ G do
    if visit[i] = 0 then
      visit[i] = 1
      if HamCycle(i) = true then
        return true
      visit[i] = 0
  return false
```

2.5 Graph coloring

```
Function gcp(v)
  if v = N + 1 then
    return true
  for i = 1 to K do
    feasible = true
    for (v, j) ∈ G do
      if color[j] = i then
        feasible = false
```

```
        break
    if feasible = true then
      color[v] = i
      if gcp(v + 1) = true then
        return true
      color[v] = 0
  return false
```

2.6 Shortest Hamiltonian path

```
Function ShortPath(v, len)
  if len >= best then
    return
  if sum(visit) = N and len < best then
    best = len
    return
  for (v, i) ∈ G do
    if visit[i] = 0 then
      visit[i] = 1
      ShortPath(i, len + M[v][i])
      visit[i] = 0
```

3 Dynamic Programming

3.1 Longest Increasing Subsequence - Top-down

```
Function lis(S, i)
  if LIS[i] is cached then
    return LIS[i]
  if i = 1 then
    LIS[i] = 1
  else
    LIS[i] = 0
  for j = 1 to i − 1 do
    LIS[j] = lis(S, j)
    if S[j] < S[i] and LIS[j] > LIS[i] then
      LIS[i] = LIS[j]
  LIS[i] = LIS[i] + 1
  return LIS[i]
```

3.2 Longest Increasing Subsequence - Bottom-up

```
Function lis(S)
  LIS[1] = 1
  for i = 2 to n do
```

```
    LIS[i] = 0
    for j = 1 to i - 1 do
      if S[j] < S[i] and LIS[j] > LIS[i] then
        LIS[i] = LIS[j]
    LIS[i] = LIS[i] + 1
  return max(LIS[1],...,LIS[n])
```

3.3 Longest Common Subsequence - Top-down

```
Function lcs(A[1..i], B[1..j])
  if LCS[i, j] is cached then
    return LCS[i, j]
  if i =0 or j =0 then
    LCS[i, j] = 0
    return LCS[i, j]
  if A[i] = B[j] then
    LCS[i, j] = lcs(A[1..i - 1], B[1..j - 1]) + 1
  else
    LCS1 = lcs(A[1..i-1], B[1..j])
    LCS2 =lcs(A[1..i], B[1..j-1])
    LCS[i, j] = max(LCS1,LCS2)
  return LCS[i, j]
```

3.4 Longest Common Subsequence - Bottom-up

```
Function lcs(A, B)
  for i = 0 to n do
    LCS[i, 0] = 0
  for j = 0 to m do
    LCS[0, j] = 0
  for i = 1 to n do
    for j = 1 to m do
      if A[i] = B[j] then
        LCS[i, j] = LCS[i - 1, j - 1] + 1
      else
        LCS[i, j] = max(LCS[i - 1, j], LCS[i, j - 1])
  return LCS[n, m]
```

3.5 Number of monotonic paths - Top-down

```
Function count(x, y)
  if T[x, y] is cached then
    return T[x, y]
  if x = 1 or y = 1 then
    return 1
  C1 = count (x - 1, y )
```

```
  C2 = count (x, y - 1)
  T[x, y] = C1 + C2
  return T[x, y]
```

3.6 Number of monotonic paths - Bottom-up

```
Function count(n, m)
  for i = 1 to n do
    T[i, 1] = 1
  for j =1 to m do
    T[1, j] = 1
  for i = 2 to n do
    for j = 2 to m do
      T[i, j] = T[i - 1, j] + T[i, j - 1])
  return T[n, m]
```

3.7 Coin Changing - Top-down

```
Function change(i, C)
  if C < 0 or i = 0 then
    return ∞
  if C = 0 then
    return 0
  if T[i, C] > 0 then
    return T[i, C]
  if di > C then
    T [i , C ] = change (i - 1, C )
  else
    T[i,C] = min(change(i - 1,C), 1+change(i,C - di))
  return T[i,C]
```

3.8 Coin Changing - Bottom-up

```
Function change(n,C)
  for i = 0 to n do
    T[i,0] = 0
  for j = 0 to C do
    T[0,j] = ∞
  for i = 1 to n do
    for j = 1 to C do
      if d_i > j then
        T[i, j] = T [i - 1, j]
      else
        T[i, j] = min(T[i - 1, j], 1 + T[i, j - d_i])
  return T[n, C]
```

3.9 Subset Sum - Top-down

```
Function subset(i, C)
  if C = 0 then
    return true
  if i = 0 and C ≠ 0 then
    return false
  if T[i, C] is not empty then
    return T[i,C]
  if d_i > C then
    T [i, C] = subset (i − 1, C )
  else
    T[i, C] = subset(i − 1, C) ∨ subset(i − 1, C − d_i)
  return T[i, C]
```

3.10 Subset Sum - Bottom-up

```
Function subset(n, C)
  for i = 0 to n do
    T[i, 0] = true
  for j = 1 to C do
    T[0, j] = false
  for i = 1 to n do
    for j = 1 to C do
      if d_i > j then
        T[i, j] = T[i − 1, j]
      else
        T[i, j] = T[i − 1, j] ∨ T[i − 1, j − d_i]
  return T[n, C]
```

3.11 Knapsack - Top-down

```
Function knapsack(i, W)
  if i = 0 then
    return 0
  if T[i, W] >= 0 then
    return T[i, W]
  if w_i > W then
    T [i, W] = knapsack (i − 1, W)
  else
    T[i, W] = max(knapsack(i − 1, W), v_i + knapsack(i − 1, W − w_i))
  return T[i, W]
```

3.12 Knapsack - Bottom-up

```
Function knapsack(n, W)
  for j =1 to W do
    T[0, j] = 0
  for i = 0 to n do
    T[i, 0] = 0
  for i = 1 to n do
    for j = 1 to W do
      if w_i > j then
        T[i, j] = T[i - 1, j]
      else
        T[i, j] = max(T[i - 1, j], v_i + T[i - 1, j - w_i])
  return T[n, W]
```

3.13 Matrix-chain multiplication - Top-down

```
Function mult(i, j)
  if j <= i then
    return 0
  if M[i, j] >= 0 then
    return M[i, j]
  cost = ∞
  for k = i to j - 1 do
    cost = min(cost, mult(i, k) + mult(k + 1, j) + p_{i-1} p_k p_j)
   M[i, j] = cost
   return cost
```

3.14 Matrix-chain multiplication - Bottom-up

```
Function mc(n)
  for d = 2 to n do
    for i = 1 to n - d + 1 do
      j = i + d - 1
      M[i, j] = ∞
      for k = i to j - 1 do
        M [i, j] = min(M[i, j], M[i, k] + M[k + 1, j] + p_{i-1} p_k p_j)
  return m[1][n]
```

4 Mooshak Problems

4.1 Radical Winter Games

```
def read_input():
    tests = []
    n_tests = input()
```

```
    for i in range(n_tests):
        n_lines = input()
        test = []
        for j in range(n_lines):
            line = raw_input()
            l = map(int,line.split(" "))
            test.append(l)
        tests.append(test)
    return tests

def calc_scores(tests):
    j=1
    for i in range(len(tests)):
        tests[i][1][0]+=tests[i][0][0]
        tests[i][1][1]+=tests[i][0][0]
        for j in range(2,len(tests[i])):
            for k in range(len(tests[i][j])):
                if k==0:
                    tests[i][j][k]+=tests[i][j-1][k]
                elif k==(len(tests[i][j])-1):
                    tests[i][j][k]+=tests[i][j-1][k-1]
                else:
                    tests[i][j][k] = max(tests[i][j][k]+tests[i][j-1][k-1],
    tests[i][j][k]+tests[i][j-1][k])
        print max(tests[i][j])

if __name__ == '__main__':
    tests = read_input()
    calc_scores(tests)
```

4.2 Train Sorting

```
#include <stdio.h>
#include <stdlib.h>

int main(){
  int n,i,j,k,l,size_train,max;
  unsigned int cars[5001];
  int aux[5001];
  scanf("%d",&n);
  for (i=0;i<n;i++){
    scanf("%d",&cars[i]);
  }
  aux[0] = 1;
  for (j=1;j<n;j++){
    size_train = 0;
    for (k=j-1;k>=0;k--){
      if (cars[j] > cars[k] && size_train < aux[k]){
        size_train = aux[k];
      }
```

```
    }
    aux[j] = ++size_train;
  }
  max = 0;
  for (l=0;l<n;l++){
    if (aux[l] > max){
      max = aux[l];
    }
  }
  printf("%d\n",max);
  return 0;
}
```

4.3 Dividing Coins

```
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <string.h>

using namespace std;

int sum = 0;
vector<int> coins;

int knapsack(int n, int w){
  int table[n+1][w+1];
  for(int j = 1; j <= w; j++){
    table[0][j] = 0;
  }
  for(int i = 0; i <= n; i++){
    table[i][0] = 0;
  }
  for(int i = 1; i <= n; i++){
    for(int j = 1; j <= w; j++){
      if(coins[i] > j){
        table[i][j] = table[i-1][j];
      }else {
        table[i][j] = max(table[i-1][j], coins[i] + table[i-1][j-coins[i]]);
      }
    }
  }
  return table[n][w];
}

int main(){
  int n_problems, n_coins, aux = 0;
  scanf("%d", &n_problems);
  for(int i = 0; i < n_problems; i++){
    sum = 0;
```

```
    coins.clear();
    scanf("%d", &n_coins);
    for(int j = 0; j < n_coins; j++){
      scanf("%d", &aux);
      coins.push_back(aux);
      sum += aux;
    }
    printf("%d\n", sum - 2*knapsack(n_coins, (int)sum/2));
  }
}
```

4.4 Two Towers

```
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <string.h>

using namespace std;

vector<int> tiles_tower_1;
vector<int> tiles_tower_2;

int lcs(int n1, int n2){
  int table[n1+1][n2+1];
  for(int i = 0; i <= n1; i++){
    table[i][0] = 0;

  }
  for(int j = 0; j <= n2; j++){
    table[0][j] = 0;

  }
  for(int i = 1; i <= n1; i++){
    for(int j = 1; j <= n2; j++){
      if(tiles_tower_1[i-1] == tiles_tower_2[j-1]){
        table[i][j] = table[i-1][j-1] + 1;
      }else {
        table[i][j] = max(table[i-1][j], table[i][j-1]);
      }
    }
  }
  return table[n1][n2];
}

int main(){
  int n1, n2, aux, k = 0;
  while(scanf("%d %d", &n1, &n2) == 2){
    if(n1 == 0 && n2 == 0){
      return 0;
```

```
    }
    k++;
    tiles_tower_2.clear();
    tiles_tower_1.clear();
    for(int i = 0; i < n1; i++){
      scanf("%d", &aux);
      tiles_tower_1.push_back(aux);
    }
    for(int i = 0; i < n2; i++){
      scanf("%d", &aux);
      tiles_tower_2.push_back(aux);
    }
    int result = lcs(n1, n2);
    printf("Twin Towers #%d\n", k);
    printf("Number of Tiles : %d\n\n", result);
  }
  return 0;
}
```

4.5 Little Red Riding Hood

```
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <string.h>

using namespace std;

unsigned long wolves[101][101];
unsigned long grid[101][101];

unsigned long count(unsigned long w, unsigned long h){
  unsigned long table[w+1][h+1];

  for(unsigned long i = 0; i < w; i++){
    table[i][0] = 1;
  }

  for(unsigned long j = 0; j < h; j++){
    table[0][j] = 1;
  }

  for(unsigned long i=0; i < w; i++){
    for(unsigned long j=0; j < h; j++){
      if(wolves[i][j] == 1){
        table[i][j] = 0;
      } else {
        if(j==0 && i==0){
          table[i][j] = 1;
        }else if(i==0){
```

```
            table[i][j] = table[i][j-1];
        }else if(j==0){
            table[i][j] = table[i-1][j];
        } else{
            table[i][j] = table[i-1][j] + table[i][j-1];
        }
      }
    }
  }
  return table[w-1][h-1];
}

int main(){
  unsigned long w, h, n_wolves, wolf_w, wolf_h, result = 0;

  while(scanf("%ld %ld", &w, &h) == 2 && (w || h)){
    for(unsigned long i = 0; i < w; i++){
      for(unsigned long j = 0; j < h; j++){
        wolves[i][j] = 0;
        grid[i][j] = 0;
      }
    }
    scanf("%ld", &n_wolves);
    for(unsigned long i = 0; i < n_wolves; i++){
      scanf("%ld %ld", &wolf_w, &wolf_h);
      wolves[wolf_w][wolf_h] = 1; // posicoes onde estao lobos
    }
    w++;
    h++;
    result = count(w,h);
    if(result == 0){
      printf("There is no path.\n");
    }else if(result == 1){
      printf("There is one path from Little Red Riding Hood's house to her
    grandmother's house.\n");
    } else {
      printf("There are %lu paths from Little Red Riding Hood's house to her
    grandmother's house.\n", result);
    }
  }
  return 0;
}
```

4.6 The Trip of Mr. Rowan

```
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <string.h>
#include <math.h>
```

```
using namespace std;

int visited_nodes[12];
int nodes[12][2];
int number_of_nodes;

double best;

void solve(int i, double distance){

  int number_visited_nodes = 0;

  double calc = 0;
  if(distance == 0)
    visited_nodes[i] = 1;

  for(int j = 0; j < number_of_nodes; j++){
    if(visited_nodes[j] == 0){
      visited_nodes[j] = 1;
      calc = sqrt(pow((nodes[j][0] - nodes[i][0]),2) + pow((nodes[j][1] -
   nodes[i][1]),2));
      distance += calc;
      if(distance <= best){
        solve(j, distance);
      }
      distance -= calc;
      visited_nodes[j] = 0;
    } else {
      number_visited_nodes++;
    }
  }
  if(number_visited_nodes == number_of_nodes){
    if(best == 0){
      best = distance;
    }
    else if(distance < best){
      best= distance;
    }
  }
  return;
}

int main(){
  int a, b = 0;
  best = 100000;
  memset(visited_nodes,0,sizeof(visited_nodes));

  scanf("%d", &number_of_nodes);
  for(int i = 0; i < number_of_nodes; i++){
    scanf("%d %d", &a, &b);
```

```
    nodes[i][0] = a;
    nodes[i][1] = b;
  }

  for(int i = 1; i < number_of_nodes; i++){
    visited_nodes[i-1] = 0;
    solve(i, 0);
  }
  printf("%.3f\n", best);
  return 0;
}
```

4.7 Delicious Pasta

```
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <string.h>
#include <math.h>

using namespace std;

int id, number_of_pastas, budget, cooking_time, taste_value = 0;
int pastas[5000][2];

double get_max_taste(){
  double cache[number_of_pastas+1][budget+1];
  int i,j,b;

  for(j = 0; j <= budget; j++){
    cache[0][j] = 0;
  }

  for(j = 0; j <= number_of_pastas; j++){
    cache[j][0] = 0;
  }

  for(i = 1; i <= number_of_pastas; i++){
    for(j = 1; j <= budget; j++){
      int inspetormax= 0;
      for(b = 1; b <= j; b++){
        inspetormax = max(cache[i-1][j], pastas[budget*(i-1) + b - 1][1] +
   cache[i-1][j-b]);
        if(inspetormax > cache[i][j]){
          cache[i][j] = inspetormax;
        }
      }
    }
  }
  return cache[number_of_pastas][budget];
```

```
}

int main() {
  int result = 0;

  scanf("%d %d", &number_of_pastas, &budget);
  for(int i = 0; i < number_of_pastas; i++){
    for(int j = 0; j < budget; j++){
      scanf("%d %d %d", &id, &cooking_time, &taste_value);
      pastas[budget*i + j%budget][0] = cooking_time;
      pastas[budget*i + j%budget][1] = taste_value;
    }
  }
  result = get_max_taste();
  printf("%d\n", result);
  return 0;
}
```

5 C++ Reference

5.1 vector

```
#include <vector>
```

- begin() - Return iterator to beginning
- end() - Return iterator to end
- size() - Return size
- empty() - Test whether vector is empty
- push_back(element) - Add element at the end
- pop_back() - Delete last element
- insert(position, val); - Insert elements
- erase(position) | erase(first, last) - Erase elements

5.2 qsort

```
void qsort(void* base, size_t num, size_t size, int (*compar)(const void*,
    const void*));
```

- base - Pointer to the first object of the array to be sorted, converted to a void*
- num - Number of elements in the array pointed to by base.
- size - Size in bytes of each element in the array.
- compar - Pointer to a function that compares two elements:

(return value - meaning)

<0 - The element pointed to by p1 goes before the element pointed to by p2

0 - The element pointed to by p1 is equivalent to the element pointed to by p2
>0 - The element pointed to by p1 goes after the element pointed to by p2
Exemplo compare:

```
int compareMyType(const void * a, const void * b){
  if ( *(MyType*)a <  *(MyType*)b ) return -1;
  if ( *(MyType*)a == *(MyType*)b ) return 0;
  if ( *(MyType*)a >  *(MyType*)b ) return 1;
}
```

Exemplo de uso:

```
#include <stdio.h>      /* printf */
#include <stdlib.h>     /* qsort */
int values[] = { 40, 10, 100, 90, 20, 25 };
int compare(const void * a, const void * b){
  return( *(int*)a - *(int*)b );
}
int main(){
  int n;
  qsort(values, 6, sizeof(int), compare);
  for (n=0; n<6; n++)
    printf("%d ",values[n]);
  return 0;
}
```