# KNU_DzaDza team notebook(2018-19)

# Contents

# 1 Combinatorial optimization

## 1.1 Sparse max-flow

```
// Adjacency list implementation of Dinic's blocking flow algorithm.
// This is very fast in practice, and only loses to push-relabel flow.
//
// Running time:
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source and sink
//
// OUTPUT:
```

```
//     - maximum flow value
//     - To obtain actual flow values, look at edges with capacity > 0
//       (zero capacity edges are residual edges).

#include<cstdio>
#include<vector>
#include<queue>
using namespace std;
typedef long long LL;

struct Edge {
  int u, v;
  LL cap, flow;
  Edge() {}
  Edge(int u, int v, LL cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic {
  int N;
  vector<Edge> E;
  vector<vector<int>> g;
  vector<int> d, pt;

  Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

  void AddEdge(int u, int v, LL cap) {
    if (u != v) {
      E.emplace_back(u, v, cap);
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(v, u, 0);
      g[v].emplace_back(E.size() - 1);
    }
  }

  bool BFS(int S, int T) {
    queue<int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while(!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k: g[u]) {
        Edge &e = E[k];
        if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    }
    return d[T] != N + 1;
  }

  LL DFS(int u, int T, LL flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      Edge &e = E[g[u][i]];
      Edge &oe = E[g[u][i]^1];
      if (d[e.v] == d[e.u] + 1) {
        LL amt = e.cap - e.flow;
        if (flow != -1 && amt > flow) amt = flow;
        if (LL pushed = DFS(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    }
    return 0;
  }

  LL MaxFlow(int S, int T) {
    LL total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (LL flow = DFS(S, T))
        total += flow;
    }
    return total;
```

```
  }
};

// BEGIN CUT
// The following code solves SPOJ problem #4110: Fast Maximum Flow (FASTFLOW)

int main()
{
  int N, E;
  scanf("%d%d", &N, &E);
  Dinic dinic(N);
  for(int i = 0; i < E; i++)
  {
    int u, v;
    LL cap;
    scanf("%d%d%lld", &u, &v, &cap);
    dinic.AddEdge(u - 1, v - 1, cap);
    dinic.AddEdge(v - 1, u - 1, cap);
  }
  printf("%lld\n", dinic.MaxFlow(0, N - 1));
  return 0;
}

// END CUT
```

## 1.2   Min-cost max-flow

```
// Implementation of min cost max flow algorithm using adjacency
// matrix (Edmonds and Karp 1972).  This implementation keeps track of
// forward and reverse edges separately (so you can set cap[i][j] !=
// cap[j][i]).  For a regular max flow, set all edge costs to 0.
//
// Running time, O(|V|^2) cost per augmentation
//     max flow:            O(|V|^3) augmentations
//     min cost max flow:   O(|V|^4 * MAX_EDGE_COST) augmentations
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - (maximum flow value, minimum cost value)
//     - To obtain the actual flow, look at positive values only.

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
  int N;
  VVL cap, flow, cost;
  VI found;
  VL dist, pi, width;
  VPII dad;

  MinCostMaxFlow(int N) :
    N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
    found(N), dist(N), pi(N), width(N), dad(N) {}

  void AddEdge(int from, int to, L cap, L cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
```

```
  }

  void Relax(int s, int k, L cap, L cost, int dir) {
    L val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }

  L Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
      int best = -1;
      found[s] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best = k;
      }
      s = best;
    }

    for (int k = 0; k < N; k++)
      pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }

  pair<L, L> GetMaxFlow(int s, int t) {
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        } else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
};

// BEGIN CUT
// The following code solves UVA problem #10594: Data Flow

int main() {
  int N, M;

  while (scanf("%d%d", &N, &M) == 2) {
    VVL v(M, VL(3));
    for (int i = 0; i < M; i++)
      scanf("%Ld%Ld%Ld", &v[i][0], &v[i][1], &v[i][2]);
    L D, K;
    scanf("%Ld%Ld", &D, &K);

    MinCostMaxFlow mcmf(N+1);
    for (int i = 0; i < M; i++) {
      mcmf.AddEdge(int(v[i][0]), int(v[i][1]), K, v[i][2]);
      mcmf.AddEdge(int(v[i][1]), int(v[i][0]), K, v[i][2]);
    }
    mcmf.AddEdge(0, 1, D, 0);

    pair<L, L> res = mcmf.GetMaxFlow(0, N);

    if (res.first == D) {
      printf("%Ld\n", res.second);
    } else {
```

```
            printf("Impossible.\n");
        }
    }

    return 0;
}

// END CUT
```

## 1.3  Push-relabel max-flow

```
// Adjacency list implementation of FIFO push relabel maximum flow
// with the gap relabeling heuristic.  This implementation is
// significantly faster than straight Ford-Fulkerson.  It solves
// random problems with 10000 vertices and 1000000 edges in a few
// seconds, though it is possible to construct test cases that
// achieve the worst-case.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).

#include <cmath>
#include <vector>
#include <iostream>
#include <queue>

using namespace std;

typedef long long LL;

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct PushRelabel {
  int N;
  vector<vector<Edge> > G;
  vector<LL> excess;
  vector<int> dist, active, count;
  queue<int> Q;

  PushRelabel(int N) : N(N), G(N), excess(N), dist(N), active(N), count(2*N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

  void Enqueue(int v) {
    if (!active[v] && excess[v] > 0) { active[v] = true; Q.push(v); }
  }

  void Push(Edge &e) {
    int amt = int(min(excess[e.from], LL(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);
  }
```

```
  void Gap(int k) {
    for (int v = 0; v < N; v++) {
      if (dist[v] < k) continue;
      count[dist[v]]--;
      dist[v] = max(dist[v], N+1);
      count[dist[v]]++;
      Enqueue(v);
    }
  }

  void Relabel(int v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (int i = 0; i < G[v].size(); i++)
      if (G[v][i].cap - G[v][i].flow > 0)
        dist[v] = min(dist[v], dist[G[v][i].to] + 1);
    count[dist[v]]++;
    Enqueue(v);
  }

  void Discharge(int v) {
    for (int i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[v][i]);
    if (excess[v] > 0) {
      if (count[dist[v]] == 1)
        Gap(dist[v]);
      else
        Relabel(v);
    }
  }

  LL GetMaxFlow(int s, int t) {
    count[0] = N-1;
    count[N] = 1;
    dist[s] = N;
    active[s] = active[t] = true;
    for (int i = 0; i < G[s].size(); i++) {
      excess[s] += G[s][i].cap;
      Push(G[s][i]);
    }

    while (!Q.empty()) {
      int v = Q.front();
      Q.pop();
      active[v] = false;
      Discharge(v);
    }

    LL totflow = 0;
    for (int i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;
    return totflow;
  }
};

// BEGIN CUT
// The following code solves SPOJ problem #4110: Fast Maximum Flow (FASTFLOW)

int main() {
  int n, m;
  scanf("%d%d", &n, &m);

  PushRelabel pr(n);
  for (int i = 0; i < m; i++) {
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    if (a == b) continue;
    pr.AddEdge(a-1, b-1, c);
    pr.AddEdge(b-1, a-1, c);
  }
  printf("%Ld\n", pr.GetMaxFlow(0, n-1));
  return 0;
}

// END CUT
```

## 1.4  Min-cost matching

```
//////////////////////////////////////////////////////////////////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs.  In practice, it solves 1000x1000 problems in around 1
// second.
//
//   cost[i][j] = cost for pairing left node i with right node j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative.  To perform
// maximization, simply negate the cost[][] matrix.
//////////////////////////////////////////////////////////////////

#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());

  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
  }

  // construct primal solution satisfying complementary slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
        Lmate[i] = j;
        Rmate[j] = i;
        mated++;
        break;
      }
    }
  }

  VD dist(n);
  VI dad(n);
  VI seen(n);

  // repeat until primal solution is feasible
  while (mated < n) {

    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;

    // initialize Dijkstra
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++)
      dist[k] = cost[s][k] - u[s] - v[k];

    int j = 0;
    while (true) {

      // find closest
      j = -1;
      for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        if (j == -1 || dist[k] < dist[j]) j = k;
      }
      seen[j] = 1;

      // termination condition
      if (Rmate[j] == -1) break;

      // relax neighbors
      const int i = Rmate[j];
      for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
        if (dist[k] > new_dist) {
          dist[k] = new_dist;
          dad[k] = j;
        }
      }
    }

    // update dual variables
    for (int k = 0; k < n; k++) {
      if (k == j || !seen[k]) continue;
      const int i = Rmate[k];
      v[k] += dist[k] - dist[j];
      u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];

    // augment along path
    while (dad[j] >= 0) {
      const int d = dad[j];
      Rmate[j] = Rmate[d];
      Lmate[Rmate[j]] = j;
      j = d;
    }
    Rmate[j] = s;
    Lmate[s] = j;

    mated++;
  }

  double value = 0;
  for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];

  return value;
}
```

## 1.5  Max bipartite matchine

```
// This code performs maximum bipartite matching.
//
// Running time: O(|E| |V|) -- often much faster in practice
//
//   INPUT: w[i][j] = edge between row node i and column node j
//   OUTPUT: mr[i] = assignment for row node i, -1 if unassigned
//           mc[j] = assignment for column node j, -1 if unassigned
//           function returns number of matches made

#include <vector>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen) {
  for (int j = 0; j < w[i].size(); j++) {
    if (w[i][j] && !seen[j]) {
      seen[j] = true;
      if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
        mr[i] = j;
```

```
            mc[j] = i;
            return true;
          }
        }
      }
    }
    return false;
  }

  int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
    mr = VI(w.size(), -1);
    mc = VI(w[0].size(), -1);

    int ct = 0;
    for (int i = 0; i < w.size(); i++) {
      VI seen(w[0].size());
      if (FindMatch(i, w, mr, mc, seen)) ct++;
    }
    return ct;
  }
```

## 1.6   Global min-cut

```
// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//
// OUTPUT:
//     - (min cut value, nodes in half of min cut)

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
  int N = weights.size();
  VI used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    VI w = weights[0];
    VI added = used;
    int prev, last = 0;
    for (int i = 0; i < phase; i++) {
      prev = last;
      last = -1;
      for (int j = 1; j < N; j++)
        if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
      if (i == phase-1) {
        for (int j = 0; j < N; j++) weights[prev][j] += weights[last][j];
        for (int j = 0; j < N; j++) weights[j][prev] = weights[prev][j];
        used[last] = true;
        cut.push_back(last);
        if (best_weight == -1 || w[last] < best_weight) {
          best_cut = cut;
          best_weight = w[last];
        }
      } else {
        for (int j = 0; j < N; j++)
          w[j] += weights[last][j];
        added[last] = true;
      }
    }
  }
  return make_pair(best_weight, best_cut);
}
```

```
}
// BEGIN CUT
// The following code solves UVA problem #10989: Bomb, Divide and Conquer
int main() {
  int N;
  cin >> N;
  for (int i = 0; i < N; i++) {
    int n, m;
    cin >> n >> m;
    VVI weights(n, VI(n));
    for (int j = 0; j < m; j++) {
      int a, b, c;
      cin >> a >> b >> c;
      weights[a-1][b-1] = weights[b-1][a-1] = c;
    }
    pair<int, VI> res = GetMinCut(weights);
    cout << "Case #" << i+1 << ": " << res.first << endl;
  }
}
// END CUT
```

## 1.7   Graph cut inference

```
// Special-purpose {0,1} combinatorial optimization solver for
// problems of the following by a reduction to graph cuts:
//
//        minimize           sum_i  psi_i(x[i])
//   x[1]...x[n] in {0,1}       + sum_{i < j}  phi_{ij}(x[i], x[j])
//
// where
//     psi_i : {0, 1} --> R
//   phi_{ij} : {0, 1} x {0, 1} --> R
//
// such that
//   phi_{ij}(0,0) + phi_{ij}(1,1) <= phi_{ij}(0,1) + phi_{ij}(1,0)   (*)
//
// This can also be used to solve maximization problems where the
// direction of the inequality in (*) is reversed.
//
// INPUT: phi -- a matrix such that phi[i][j][u][v] = phi_{ij}(u, v)
//        psi -- a matrix such that psi[i][u] = psi_i(u)
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution
//
// To use this code, create a GraphCutInference object, and call the
// DoInference() method.  To perform maximization instead of minimization,
// ensure that #define MAXIMIZATION is enabled.

#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef vector<VVI> VVVI;
typedef vector<VVVI> VVVVI;

const int INF = 1000000000;

// comment out following line for minimization
#define MAXIMIZATION

struct GraphCutInference {
  int N;
  VVI cap, flow;
  VI reached;

  int Augment(int s, int t, int a) {
    reached[s] = 1;
    if (s == t) return a;
    for (int k = 0; k < N; k++) {
      if (reached[k]) continue;
```

```
      if (int aa = min(a, cap[s][k] - flow[s][k])) {
        if (int b = Augment(k, t, aa)) {
          flow[s][k] += b;
          flow[k][s] -= b;
          return b;
        }
      }
    }
    return 0;
  }

  int GetMaxFlow(int s, int t) {
    N = cap.size();
    flow = VVI(N, VI(N));
    reached = VI(N);

    int totflow = 0;
    while (int amt = Augment(s, t, INF)) {
      totflow += amt;
      fill(reached.begin(), reached.end(), 0);
    }
    return totflow;
  }

  int DoInference(const VVVVI &phi, const VVI &psi, VI &x) {
    int M = phi.size();
    cap = VVI(M+2, VI(M+2));
    VI b(M);
    int c = 0;

    for (int i = 0; i < M; i++) {
      b[i] += psi[i][1] - psi[i][0];
      c += psi[i][0];
      for (int j = 0; j < i; j++)
        b[i] += phi[i][j][1][1] - phi[i][j][0][1];
      for (int j = i+1; j < M; j++) {
        cap[i][j] = phi[i][j][0][1] + phi[i][j][1][0] - phi[i][j][0][0] - phi[i
            ][j][1][1];
        b[i] += phi[i][j][1][0] - phi[i][j][0][0];
        c += phi[i][j][0][0];
      }
    }

#ifdef MAXIMIZATION
    for (int i = 0; i < M; i++) {
      for (int j = i+1; j < M; j++)
        cap[i][j] *= -1;
      b[i] *= -1;
    }
    c *= -1;
#endif

    for (int i = 0; i < M; i++) {
      if (b[i] >= 0) {
        cap[M][i] = b[i];
      } else {
        cap[i][M+1] = -b[i];
        c += b[i];
      }
    }

    int score = GetMaxFlow(M, M+1);
    fill(reached.begin(), reached.end(), 0);
    Augment(M, M+1, INF);
    x = VI(M);
    for (int i = 0; i < M; i++) x[i] = reached[i] ? 0 : 1;
    score += c;
#ifdef MAXIMIZATION
    score *= -1;
#endif

    return score;
  }
};

int main() {

  // solver for "Cat vs. Dog" from NWERC 2008
```

```
  int numcases;
  cin >> numcases;
  for (int caseno = 0; caseno < numcases; caseno++) {
    int c, d, v;
    cin >> c >> d >> v;

    VVVVI phi(c+d, VVVI(c+d, VVI(2, VI(2))));
    VVI psi(c+d, VI(2));
    for (int i = 0; i < v; i++) {
      char p, q;
      int u, v;
      cin >> p >> u >> q >> v;
      u--; v--;
      if (p == 'C') {
        phi[u][c+v][0][0]++;
        phi[c+v][u][0][0]++;
      } else {
        phi[v][c+u][1][1]++;
        phi[c+u][v][1][1]++;
      }
    }

    GraphCutInference graph;
    VI x;
    cout << graph.DoInference(phi, psi, x) << endl;
  }

  return 0;
}
```

## 1.8   Min-cost max-flow (Java)

```
// Min cost max flow algorithm using an adjacency matrix.  If you
// want just regular max flow, setting all edge costs to 1 gives
// running time O(|E|^2 |V|).
//
// Running time: O(min(|V|^2 * totflow, |V|^3 * totcost))
//
// INPUT: cap -- a matrix such that cap[i][j] is the capacity of
//             a directed edge from node i to node j
//
//        cost -- a matrix such that cost[i][j] is the (positive)
//             cost of sending one unit of flow along a
//             directed edge from node i to node j
//
//        source -- starting node
//        sink -- ending node
//
// OUTPUT: max flow and min cost; the matrix flow will contain
//        the actual flow values (note that unlike in the MaxFlow
//        code, you don't need to ignore negative flow values --
//        there
//        shouldn't be any)
//
// To use this, create a MinCostMaxFlow object, and call it like this:
//
//   MinCostMaxFlow nf;
//   int maxflow = nf.getMaxFlow(cap,cost,source,sink);

import java.util.*;

public class MinCostMaxFlow {
    boolean found[];
    int N, cap[][], flow[][], cost[][], dad[], dist[], pi[];

    static final int INF = Integer.MAX_VALUE / 2 - 1;

    boolean search(int source, int sink) {
        Arrays.fill(found, false);
        Arrays.fill(dist, INF);
        dist[source] = 0;

        while (source != N) {
            int best = N;
            found[source] = true;
```

```
        for (int k = 0; k < N; k++) {
            if (found[k]) continue;
            if (flow[k][source] != 0) {
                int val = dist[source] + pi[source] - pi[k] - cost[k][source
                    ];
                if (dist[k] > val) {
                    dist[k] = val;
                    dad[k] = source;
                }
            }
            if (flow[source][k] < cap[source][k]) {
                int val = dist[source] + pi[source] - pi[k] + cost[source][k
                    ];
                if (dist[k] > val) {
                    dist[k] = val;
                    dad[k] = source;
                }
            }

            if (dist[k] < dist[best]) best = k;
        }
        source = best;
    }
    for (int k = 0; k < N; k++)
        pi[k] = Math.min(pi[k] + dist[k], INF);
    return found[sink];
}


int[] getMaxFlow(int cap[][], int cost[][], int source, int sink) {
    this.cap = cap;
    this.cost = cost;

    N = cap.length;
    found = new boolean[N];
    flow = new int[N][N];
    dist = new int[N+1];
    dad = new int[N];
    pi = new int[N];

    int totflow = 0, totcost = 0;
    while (search(source, sink)) {
        int amt = INF;
        for (int x = sink; x != source; x = dad[x])
            amt = Math.min(amt, flow[x][dad[x]] != 0 ? flow[x][dad[x]] :
                    cap[dad[x]][x] - flow[dad[x]][x]);
        for (int x = sink; x != source; x = dad[x]) {
            if (flow[x][dad[x]] != 0) {
                flow[x][dad[x]] -= amt;
                totcost -= amt * cost[x][dad[x]];
            } else {
                flow[dad[x]][x] += amt;
                totcost += amt * cost[dad[x]][x];
            }
        }
        totflow += amt;
    }

    return new int[]{ totflow, totcost };
}

public static void main (String args[]){
    MinCostMaxFlow flow = new MinCostMaxFlow();
    int cap[][] = {{0, 3, 4, 5, 0},
                   {0, 0, 2, 0, 0},
                   {0, 0, 0, 4, 1},
                   {0, 0, 0, 0, 10},
                   {0, 0, 0, 0, 0}};

    int cost1[][] = {{0, 1, 0, 0, 0},
                     {0, 0, 0, 0, 0},
                     {0, 0, 0, 0, 0},
                     {0, 0, 0, 0, 0},
                     {0, 0, 0, 0, 0}};

    int cost2[][] = {{0, 0, 1, 0, 0},
                     {0, 0, 0, 0, 0},
                     {0, 0, 0, 0, 0},
                     {0, 0, 0, 0, 0},
```

```
                     {0, 0, 0, 0, 0}};

    // should print out:
    //    10 1
    //    10 3

    int ret1[] = flow.getMaxFlow(cap, cost1, 0, 4);
    int ret2[] = flow.getMaxFlow(cap, cost2, 0, 4);

    System.out.println (ret1[0] + " " + ret1[1]);
    System.out.println (ret2[0] + " " + ret2[1]);
    }
}
```

# 2 Geometry

## 2.1 Convex hull

```
// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm.  Eliminate redundant points from the hull if REMOVE_REDUNDANT is
// #defined.
//
// Running time: O(n log n)
//
//   INPUT:   a vector of input points, unordered.
//   OUTPUT:  a vector of points in the convex hull, counterclockwise, starting
//            with bottommost/leftmost point

#include <cstdio>
#include <cassert>
#include <vector>
#include <algorithm>
#include <cmath>
// BEGIN CUT
#include <map>
// END CUT

using namespace std;

#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs) const { return make_pair(y,x) < make_pair(rhs.y,
      rhs.x); }
  bool operator==(const PT &rhs) const { return make_pair(y,x) == make_pair(rhs.
      y,rhs.x); }
};

T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) + cross(c,a); }

#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
  return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y
      -b.y) <= 0);
}
#endif

void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(), pts.end()), pts.end());
  vector<PT> up, dn;
  for (int i = 0; i < pts.size(); i++) {
    while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.
        pop_back();
    while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.
        pop_back();
    up.push_back(pts[i]);
    dn.push_back(pts[i]);
```

```
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
      if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
      dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
      dn[0] = dn.back();
      dn.pop_back();
    }
    pts = dn;
#endif
}

// BEGIN CUT
// The following code solves SPOJ problem #26: Build the Fence (BSHEEP)

int main() {
  int t;
  scanf("%d", &t);
  for (int caseno = 0; caseno < t; caseno++) {
    int n;
    scanf("%d", &n);
    vector<PT> v(n);
    for (int i = 0; i < n; i++) scanf("%lf%lf", &v[i].x, &v[i].y);
    vector<PT> h(v);
    map<PT,int> index;
    for (int i = n-1; i >= 0; i--) index[v[i]] = i+1;
    ConvexHull(h);

    double len = 0;
    for (int i = 0; i < h.size(); i++) {
      double dx = h[i].x - h[(i+1)%h.size()].x;
      double dy = h[i].y - h[(i+1)%h.size()].y;
      len += sqrt(dx*dx+dy*dy);
    }

    if (caseno > 0) printf("\n");
    printf("%.2f\n", len);
    for (int i = 0; i < h.size(); i++) {
      if (i > 0) printf(" ");
      printf("%d", index[h[i]]);
    }
    printf("\n");
  }
}

// END CUT
```

## 2.2 Miscellaneous geometry

```
// C++ routines for computational geometry.

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c  ); }
  PT operator / (double c)     const { return PT(x/c,   y/c  ); }
};

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  return os << "(" << p.x << "," << p.y << ")";
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
        dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
```

```cpp
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}

int main() {

  // expected: (-5,2)
  cerr << RotateCCW90(PT(2,5)) << endl;

  // expected: (5,-2)
  cerr << RotateCW90(PT(2,5)) << endl;

  // expected: (-5,2)
  cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

  // expected: (5,2)
  cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

  // expected: (5,2) (7.5,3) (2.5,1)
  cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
       << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
       << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

  // expected: 6.78903
  cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

  // expected: 1 0 1
  cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 0 0 1
  cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 1 1 1 0
  cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

  // expected: (1,2)
  cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << endl;

  // expected: (1,1)
  cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;
```

```cpp
    vector<PT> v;
    v.push_back(PT(0,0));
    v.push_back(PT(5,0));
    v.push_back(PT(5,5));
    v.push_back(PT(0,5));

    // expected: 1 1 1 0 0
    cerr << PointInPolygon(v, PT(2,2)) << " "
        << PointInPolygon(v, PT(2,0)) << " "
        << PointInPolygon(v, PT(0,2)) << " "
        << PointInPolygon(v, PT(5,2)) << " "
        << PointInPolygon(v, PT(2,5)) << endl;

    // expected: 0 1 1 1 1
    cerr << PointOnPolygon(v, PT(2,2)) << " "
        << PointOnPolygon(v, PT(2,0)) << " "
        << PointOnPolygon(v, PT(0,2)) << " "
        << PointOnPolygon(v, PT(5,2)) << " "
        << PointOnPolygon(v, PT(2,5)) << endl;

    // expected: (1,6)
    //           (5,4) (4,5)
    //           blank line
    //           (4,5) (5,4)
    //           blank line
    //           (4,5) (5,4)
    vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
    for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
    u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
    for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
    u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
    for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
    u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
    for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
    u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
    for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
    u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
    for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;

    // area should be 5.0
    // centroid should be (1.1666666, 1.166666)
    PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
    vector<PT> p(pa, pa+4);
    PT c = ComputeCentroid(p);
    cerr << "Area: " << ComputeArea(p) << endl;
    cerr << "Centroid: " << c << endl;

    return 0;
}
```

## 2.3  Slow Delaunay triangulation

```cpp
// Slow but simple Delaunay triangulation. Does not handle
// degenerate cases (from O'Rourke, Computational Geometry in C)
//
// Running time: O(n^4)
//
// INPUT:    x[] = x-coordinates
//           y[] = y-coordinates
//
// OUTPUT:   triples = a vector containing m triples of indices
//                     corresponding to triangle vertices

#include<vector>
using namespace std;

typedef double T;

struct triple {
    int i, j, k;
    triple() {}
    triple(int i, int j, int k) : i(i), j(j), k(k) {}
};
```

```cpp
vector<triple> delaunayTriangulation(vector<T>& x, vector<T>& y) {
    int n = x.size();
    vector<T> z(n);
    vector<triple> ret;

    for (int i = 0; i < n; i++)
        z[i] = x[i] * x[i] + y[i] * y[i];

    for (int i = 0; i < n-2; i++) {
        for (int j = i+1; j < n; j++) {
            for (int k = i+1; k < n; k++) {
                if (j == k) continue;
                double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j]-z[i
                    ]);
                double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k]-z[i
                    ]);
                double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j]-y[i
                    ]);
                bool flag = zn < 0;
                for (int m = 0; flag && m < n; m++)
                    flag = flag && ((x[m]-x[i])*xn +
                                    (y[m]-y[i])*yn +
                                    (z[m]-z[i])*zn <= 0);
                if (flag) ret.push_back(triple(i, j, k));
            }
        }
    }
    return ret;
}

int main()
{
    T xs[]={0, 0, 1, 0.9};
    T ys[]={0, 1, 0, 0.9};
    vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
    vector<triple> tri = delaunayTriangulation(x, y);

    //expected: 0 1 3
    //          0 3 2

    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

# 3  Numerical algorithms

## 3.1  Number theory (modular, Chinese remainder, linear Diophantine)

```cpp
// This is a collection of useful code for solving problems that
// involve modular linear equations.  Note that all of the
// algorithms described here work on nonnegative integers.

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> VI;
typedef pair<int, int> PII;

// return a % b (positive value)
int mod(int a, int b) {
    return ((a%b) + b) % b;
}

// computes gcd(a,b)
int gcd(int a, int b) {
    while (b) { int t = a%b; a = b; b = t; }
```

```cpp
        return a;
}

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

// (a^b) mod m via successive squaring
int powermod(int a, int b, int m)
{
        int ret = 1;
        while (b)
        {
                if (b & 1) ret = mod(ret*a, m);
                a = mod(a*a, m);
                b >>= 1;
        }
        return ret;
}

// returns g = gcd(a, b); finds x, y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
        int xx = y = 0;
        int yy = x = 1;
        while (b) {
                int q = a / b;
                int t = b; b = a%b; a = t;
                t = xx; xx = x - q*xx; x = t;
                t = yy; yy = y - q*yy; y = t;
        }
        return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
        int x, y;
        VI ret;
        int g = extended_euclid(a, n, x, y);
        if (!(b%g)) {
                x = mod(x*(b / g), n);
                for (int i = 0; i < g; i++)
                        ret.push_back(mod(x + i*(n / g), n));
        }
        return ret;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
        int x, y;
        int g = extended_euclid(a, n, x, y);
        if (g > 1) return -1;
        return mod(x, n);
}

// Chinese remainder theorem (special case): find z such that
// z % m1 = r1, z % m2 = r2.  Here, z is unique modulo M = lcm(m1, m2).
// Return (z, M).  On failure, M = -1.
PII chinese_remainder_theorem(int m1, int r1, int m2, int r2) {
        int s, t;
        int g = extended_euclid(m1, m2, s, t);
        if (r1%g != r2%g) return make_pair(0, -1);
        return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2) / g, m1*m2 / g);
}

// Chinese remainder theorem: find z such that
// z % m[i] = r[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (m[i]).  Return (z, M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &m, const VI &r) {
        PII ret = make_pair(r[0], m[0]);
        for (int i = 1; i < m.size(); i++) {
                ret = chinese_remainder_theorem(ret.second, ret.first, m[i], r[i
                        ]);
                if (ret.second == -1) break;
        }
        return ret;
}
```

```cpp
// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x, int &y) {
        if (!a && !b)
        {
                if (c) return false;
                x = 0; y = 0;
                return true;
        }
        if (!a)
        {
                if (c % b) return false;
                x = 0; y = c / b;
                return true;
        }
        if (!b)
        {
                if (c % a) return false;
                x = c / a; y = 0;
                return true;
        }
        int g = gcd(a, b);
        if (c % g) return false;
        x = c / g * mod_inverse(a / g, b / g);
        y = (c - a*x) / b;
        return true;
}

int main() {
        // expected: 2
        cout << gcd(14, 30) << endl;

        // expected: 2 -2 1
        int x, y;
        int g = extended_euclid(14, 30, x, y);
        cout << g << " " << x << " " << y << endl;

        // expected: 95 451
        VI sols = modular_linear_equation_solver(14, 30, 100);
        for (int i = 0; i < sols.size(); i++) cout << sols[i] << " ";
        cout << endl;

        // expected: 8
        cout << mod_inverse(8, 9) << endl;

        // expected: 23 105
        //           11 12
        PII ret = chinese_remainder_theorem(VI({ 3, 5, 7 }), VI({ 2, 3, 2 }));
        cout << ret.first << " " << ret.second << endl;
        ret = chinese_remainder_theorem(VI({ 4, 6 }), VI({ 3, 5 }));
        cout << ret.first << " " << ret.second << endl;

        // expected: 5 -15
        if (!linear_diophantine(7, 2, 5, x, y)) cout << "ERROR" << endl;
        cout << x << " " << y << endl;
        return 0;
}
```

## 3.2  Systems of linear equations, matrix inverse, determinant

```cpp
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//    (1) solving systems of linear equations (AX=B)
//    (2) inverting matrices (AX=I)
//    (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
// OUTPUT:   X     = an nxm matrix (stored in b[][])
```

```
//             A^{-1} = an nxn matrix (stored in a[][])
//             returns determinant of a[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;

  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
      for (int k = 0; k < n; k++) if (!ipiv[k])
        if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0);
          }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj;
    icol[i] = pk;

    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p] *= c;
    for (int p = 0; p < m; p++) b[pk][p] *= c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
    }
  }

  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
  }

  return det;
}

int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
  double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
  VVT a(n), b(n);
  for (int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }

  double det = GaussJordan(a, b);

  // expected: 60
  cout << "Determinant: " << det << endl;

  // expected: -0.233333 0.166667 0.133333 0.0666667
  //            0.166667 0.166667 0.333333 -0.333333
  //            0.233333 0.833333 -0.133333 -0.0666667
  //            0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

  // expected: 1.63333 1.3
  //            -0.166667 0.5
  //            2.36667 1.7
  //            -1.85 -1.35
  cout << "Solution: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

## 3.3   Reduced row echelon form, matrix rank

```
// Reduced row echelon form via Gauss-Jordan elimination
// with partial pivoting.  This can be used for computing
// the rank of a matrix.
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxm matrix
//
// OUTPUT:   rref[][] = an nxm matrix (stored in a[][])
//           returns rank of a[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPSILON = 1e-10;

typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

int rref(VVT &a) {
  int n = a.size();
  int m = a[0].size();
  int r = 0;
  for (int c = 0; c < m && r < n; c++) {
    int j = r;
    for (int i = r + 1; i < n; i++)
      if (fabs(a[i][c]) > fabs(a[j][c])) j = i;
    if (fabs(a[j][c]) < EPSILON) continue;
    swap(a[j], a[r]);

    T s = 1.0 / a[r][c];
    for (int j = 0; j < m; j++) a[r][j] *= s;
    for (int i = 0; i < n; i++) if (i != r) {
      T t = a[i][c];
      for (int j = 0; j < m; j++) a[i][j] -= t * a[r][j];
    }
    r++;
  }
  return r;
}

int main() {
  const int n = 5, m = 4;
  double A[n][m] = {
    {16,  2,  3, 13},
    { 5, 11, 10,  8},
    { 9,  7,  6, 12},
    { 4, 14, 15,  1},
    {13, 21, 21, 13}};
  VVT a(n);
  for (int i = 0; i < n; i++)
    a[i] = VT(A[i], A[i] + m);
```

```cpp
  int rank = rref(a);

  // expected: 3
  cout << "Rank: " << rank << endl;

  // expected: 1 0 0 1
  //           0 1 0 3
  //           0 0 1 -3
  //           0 0 0 3.10862e-15
  //           0 0 0 2.22045e-15
  cout << "rref: " << endl;
  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 4; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }
}
```

## 3.4  Fast Fourier transform

```cpp
#include <cassert>
#include <cstdio>
#include <cmath>

struct cpx
{
  cpx(){}
  cpx(double aa):a(aa),b(0){}
  cpx(double aa, double bb):a(aa),b(bb){}
  double a;
  double b;
  double modsq(void) const
  {
    return a * a + b * b;
  }
  cpx bar(void) const
  {
    return cpx(a, -b);
  }
};

cpx operator +(cpx a, cpx b)
{
  return cpx(a.a + b.a, a.b + b.b);
}

cpx operator *(cpx a, cpx b)
{
  return cpx(a.a * b.a - a.b * b.b, a.a * b.b + a.b * b.a);
}

cpx operator /(cpx a, cpx b)
{
  cpx r = a * b.bar();
  return cpx(r.a / b.modsq(), r.b / b.modsq());
}

cpx EXP(double theta)
{
  return cpx(cos(theta),sin(theta));
}

const double two_pi = 4 * acos(0);

// in:      input array
// out:     output array
// step:    {SET TO 1} (used internally)
// size:    length of the input/output {MUST BE A POWER OF 2}
// dir:     either plus or minus one (direction of the FFT)
// RESULT: out[k] = \sum_{j=0}^{size - 1} in[j] * exp(dir * 2pi * i * j * k /
//     size)
void FFT(cpx *in, cpx *out, int step, int size, int dir)
{
  if(size < 1) return;
  if(size == 1)
```

```cpp
  {
    out[0] = in[0];
    return;
  }
  FFT(in, out, step * 2, size / 2, dir);
  FFT(in + step, out + size / 2, step * 2, size / 2, dir);
  for(int i = 0 ; i < size / 2 ; i++)
  {
    cpx even = out[i];
    cpx odd = out[i + size / 2];
    out[i] = even + EXP(dir * two_pi * i / size) * odd;
    out[i + size / 2] = even + EXP(dir * two_pi * (i + size / 2) / size) * odd;
  }
}

// Usage:
// f[0...N-1] and g[0..N-1] are numbers
// Want to compute the convolution h, defined by
// h[n] = sum of f[k]g[n-k] (k = 0, ..., N-1).
// Here, the index is cyclic; f[-1] = f[N-1], f[-2] = f[N-2], etc.
// Let F[0...N-1] be FFT(f), and similarly, define G and H.
// The convolution theorem says H[n] = F[n]G[n] (element-wise product).
// To compute h[] in O(N log N) time, do the following:
//   1. Compute F and G (pass dir = 1 as the argument).
//   2. Get H by element-wise multiplying F and G.
//   3. Get h by taking the inverse FFT (use dir = -1 as the argument)
//      and *dividing by N*. DO NOT FORGET THIS SCALING FACTOR.

int main(void)
{
  printf("If rows come in identical pairs, then everything works.\n");

  cpx a[8] = {0, 1, cpx(1,3), cpx(0,5), 1, 0, 2, 0};
  cpx b[8] = {1, cpx(0,-2), cpx(0,1), 3, -1, -3, 1, -2};
  cpx A[8];
  cpx B[8];
  FFT(a, A, 1, 8, 1);
  FFT(b, B, 1, 8, 1);

  for(int i = 0 ; i < 8 ; i++)
  {
    printf("%7.2lf%7.2lf", A[i].a, A[i].b);
  }
  printf("\n");
  for(int i = 0 ; i < 8 ; i++)
  {
    cpx Ai(0,0);
    for(int j = 0 ; j < 8 ; j++)
    {
      Ai = Ai + a[j] * EXP(j * i * two_pi / 8);
    }
    printf("%7.2lf%7.2lf", Ai.a, Ai.b);
  }
  printf("\n");

  cpx AB[8];
  for(int i = 0 ; i < 8 ; i++)
    AB[i] = A[i] * B[i];
  cpx aconvb[8];
  FFT(AB, aconvb, 1, 8, -1);
  for(int i = 0 ; i < 8 ; i++)
    aconvb[i] = aconvb[i] / 8;
  for(int i = 0 ; i < 8 ; i++)
  {
    printf("%7.2lf%7.2lf", aconvb[i].a, aconvb[i].b);
  }
  printf("\n");
  for(int i = 0 ; i < 8 ; i++)
  {
    cpx aconvbi(0,0);
    for(int j = 0 ; j < 8 ; j++)
    {
      aconvbi = aconvbi + a[j] * b[(8 + i - j) % 8];
    }
    printf("%7.2lf%7.2lf", aconvbi.a, aconvbi.b);
  }
  printf("\n");

  return 0;
```

## 3.5 Simplex algorithm

```cpp
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize       c^T x
//     subject to    Ax <= b
//                   x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;

  LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i
        ]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m + 1][n] = 1;
  }

  void Pivot(int r, int s) {
    double inv = 1.0 / D[r][s];
    for (int i = 0; i < m + 2; i++) if (i != r)
      for (int j = 0; j < n + 2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] * inv;
    for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
    for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool Simplex(int phase) {
    int x = phase == 1 ? m + 1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s]) s
            = j;
      }
      if (D[x][s] > -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
          (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] < B[r]) r =
              i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }

  DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
    if (D[r][n + 1] < -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -numeric_limits<DOUBLE
          >::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] < N[s])
              s = j;
        Pivot(i, s);
      }
    }
    if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
    return D[m][n + 1];
  }
};

int main() {

  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };

  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);

  cerr << "VALUE: " << value << endl; // VALUE: 1.29032
  cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;
}
```

# 4 Graph algorithms

## 4.1 Fast Dijkstra's algorithm

```cpp
// Implementation of Dijkstra's algorithm using adjacency lists
// and priority queue for efficiency.
//
// Running time: O(|E| log |V|)

#include <queue>
#include <cstdio>

using namespace std;
const int INF = 2000000000;
```

```cpp
typedef pair<int, int> PII;

int main() {

        int N, s, t;
        scanf("%d%d%d", &N, &s, &t);
        vector<vector<PII> > edges(N);
        for (int i = 0; i < N; i++) {
                int M;
                scanf("%d", &M);
                for (int j = 0; j < M; j++) {
                        int vertex, dist;
                        scanf("%d%d", &vertex, &dist);
                        edges[i].push_back(make_pair(dist, vertex)); // note
                                order of arguments here
                }
        }

        // use priority queue in which top element has the "smallest" priority
        priority_queue<PII, vector<PII>, greater<PII> > Q;
        vector<int> dist(N, INF), dad(N, -1);
        Q.push(make_pair(0, s));
        dist[s] = 0;
        while (!Q.empty()) {
                PII p = Q.top();
                Q.pop();
                int here = p.second;
                if (here == t) break;
                if (dist[here] != p.first) continue;

                for (vector<PII>::iterator it = edges[here].begin(); it != edges
                        [here].end(); it++) {
                        if (dist[here] + it->first < dist[it->second]) {
                                dist[it->second] = dist[here] + it->first;
                                dad[it->second] = here;
                                Q.push(make_pair(dist[it->second], it->second));
                        }
                }
        }

        printf("%d\n", dist[t]);
        if (dist[t] < INF)
                for (int i = t; i != -1; i = dad[i])
                        printf("%d%c", i, (i == s ? '\n' : ' '));
        return 0;
}

/*
Sample input:
5 0 4
2 1 2 3 1
2 2 4 4 5
3 1 4 3 3 4 1
2 0 1 2 3
2 1 5 2 1

Expected:
5
4 2 3 0
*/
```

## 4.2  Strongly connected components

```cpp
#include<memory.h>
struct edge{int e, nxt;};
int V, E;
edge e[MAXE], er[MAXE];
int sp[MAXV], spr[MAXV];
int group_cnt, group_num[MAXV];
bool v[MAXV];
int stk[MAXV];
void fill_forward(int x)
{
  int i;
  v[x]=true;
```

```cpp
  for(i=sp[x];i;i=e[i].nxt) if(!v[e[i].e]) fill_forward(e[i].e);
  stk[++stk[0]]=x;
}
void fill_backward(int x)
{
  int i;
  v[x]=false;
  group_num[x]=group_cnt;
  for(i=spr[x];i;i=er[i].nxt) if(v[er[i].e]) fill_backward(er[i].e);
}
void add_edge(int v1, int v2) //add edge v1->v2
{
  e [++E].e=v2; e [E].nxt=sp [v1]; sp [v1]=E;
  er[  E].e=v1; er[E].nxt=spr[v2]; spr[v2]=E;
}
void SCC()
{
  int i;
  stk[0]=0;
  memset(v, false, sizeof(v));
  for(i=1;i<=V;i++) if(!v[i]) fill_forward(i);
  group_cnt=0;
  for(i=stk[0];i>=1;i--) if(v[stk[i]]){group_cnt++; fill_backward(stk[i]);}
}
```

## 4.3  Eulerian path

```cpp
struct Edge;
typedef list<Edge>::iterator iter;

struct Edge
{
        int next_vertex;
        iter reverse_edge;

        Edge(int next_vertex)
                :next_vertex(next_vertex)
                { }
};

const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];              // adjacency list

vector<int> path;

void find_path(int v)
{
        while(adj[v].size() > 0)
        {
                int vn = adj[v].front().next_vertex;
                adj[vn].erase(adj[v].front().reverse_edge);
                adj[v].pop_front();
                find_path(vn);
        }
        path.push_back(v);
}

void add_edge(int a, int b)
{
        adj[a].push_front(Edge(b));
        iter ita = adj[a].begin();
        adj[b].push_front(Edge(a));
        iter itb = adj[b].begin();
        ita->reverse_edge = itb;
        itb->reverse_edge = ita;
}
```

# 5 Data structures

## 5.1 Suffix array

```cpp
// Suffix array construction in O(L log^2 L) time.  Routine for
// computing the length of the longest common prefix of any two
// suffixes in O(log L) time.
//
// INPUT:    string s
//
// OUTPUT:  array suffix[] such that suffix[i] = index (from 0 to L-1)
//          of substring s[i...L-1] in the list of sorted suffixes.
//          That is, if we take the inverse of the permutation suffix[],
//          we get the actual suffix array.

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

using namespace std;

struct SuffixArray {
  const int L;
  string s;
  vector<vector<int> > P;
  vector<pair<pair<int,int>,int> > M;

  SuffixArray(const string &s) : L(s.length()), s(s), P(1, vector<int>(L, 0)), M
      (L) {
    for (int i = 0; i < L; i++) P[0][i] = int(s[i]);
    for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
      P.push_back(vector<int>(L, 0));
      for (int i = 0; i < L; i++)
        M[i] = make_pair(make_pair(P[level-1][i], i + skip < L ? P[level-1][i +
            skip] : -1000), i);
      sort(M.begin(), M.end());
      for (int i = 0; i < L; i++)
        P[level][M[i].second] = (i > 0 && M[i].first == M[i-1].first) ? P[level
            ][M[i-1].second] : i;
    }
  }

  vector<int> GetSuffixArray() { return P.back(); }

  // returns the length of the longest common prefix of s[i...L-1] and s[j...L
      -1]
  int LongestCommonPrefix(int i, int j) {
    int len = 0;
    if (i == j) return L - i;
    for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--) {
      if (P[k][i] == P[k][j]) {
        i += 1 << k;
        j += 1 << k;
        len += 1 << k;
      }
    }
    return len;
  }
};

// BEGIN CUT
// The following code solves UVA problem 11512: GATTACA.
#define TESTING
#ifdef TESTING
int main() {
  int T;
  cin >> T;
  for (int caseno = 0; caseno < T; caseno++) {
    string s;
    cin >> s;
    SuffixArray array(s);
    vector<int> v = array.GetSuffixArray();
    int bestlen = -1, bestpos = -1, bestcount = 0;
    for (int i = 0; i < s.length(); i++) {
      int len = 0, count = 0;
      for (int j = i+1; j < s.length(); j++) {
        int l = array.LongestCommonPrefix(i, j);
        if (l >= len) {
          if (l > len) count = 2; else count++;
          len = l;
        }
      }
      if (len > bestlen || len == bestlen && s.substr(bestpos, bestlen) > s.
          substr(i, len)) {
        bestlen = len;
        bestcount = count;
        bestpos = i;
      }
    }
    if (bestlen == 0) {
      cout << "No repetitions found!" << endl;
    } else {
      cout << s.substr(bestpos, bestlen) << " " << bestcount << endl;
    }
  }
}

#else
// END CUT
int main() {

  // bobocel is the 0'th suffix
  //  obocel is the 5'th suffix
  //   bocel is the 1'st suffix
  //    ocel is the 6'th suffix
  //     cel is the 2'nd suffix
  //      el is the 3'rd suffix
  //       l is the 4'th suffix
  SuffixArray suffix("bobocel");
  vector<int> v = suffix.GetSuffixArray();

  // Expected output: 0 5 1 6 2 3 4
  //                  2
  for (int i = 0; i < v.size(); i++) cout << v[i] << " ";
  cout << endl;
  cout << suffix.LongestCommonPrefix(0, 2) << endl;

}
// BEGIN CUT
#endif
// END CUT
```

## 5.2 Binary Indexed Tree

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

#define LOGSZ 17

int tree[(1<<LOGSZ)+1];
int N = (1<<LOGSZ);

// add v to value at x
void set(int x, int v) {
  while(x <= N) {
    tree[x] += v;
    x += (x & -x);
  }
}

// get cumulative sum up to and including x
int get(int x) {
  int res = 0;
  while(x) {
    res += tree[x];
    x -= (x & -x);
  }
  return res;
}
```

```
// get largest value with cumulative sum less than or equal to x;
// for smallest, pass x-1 and add 1 to result
int getind(int x) {
  int idx = 0, mask = N;
  while(mask && idx < N) {
    int t = idx + mask;
    if(x >= tree[t]) {
      idx = t;
      x -= tree[t];
    }
    mask >>= 1;
  }
  return idx;
}
```

## 5.3  Union-find set

```
#include <iostream>
#include <vector>
using namespace std;
struct UnionFind {
    vector<int> C;
    UnionFind(int n) : C(n) { for (int i = 0; i < n; i++) C[i] = i; }
    int find(int x) { return (C[x] == x) ? x : C[x] = find(C[x]); }
    void merge(int x, int y) { C[find(x)] = find(y); }
};
int main()
{
    int n = 5;
    UnionFind uf(n);
    uf.merge(0, 2);
    uf.merge(1, 0);
    uf.merge(3, 4);
    for (int i = 0; i < n; i++) cout << i << " " << uf.find(i) << endl;
    return 0;
}
```

## 5.4  KD-tree

```
// ------------------------------------------------------------------
// A straightforward, but probably sub-optimal KD-tree implmentation
// that's probably good enough for most things (current it's a
// 2D-tree)
//
//  - constructs from n points in O(n lg^2 n) time
//  - handles nearest-neighbor query in O(lg n) if points are well
//    distributed
//  - worst case for nearest-neighbor may be linear in pathological
//    case
//
// Sonny Chan, Stanford University, April 2009
// ------------------------------------------------------------------

#include <iostream>
#include <vector>
#include <limits>
#include <cstdlib>

using namespace std;

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

bool operator==(const point &a, const point &b)
{
```

```
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b)
{
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b)
{
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b)
{
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox
{
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < v.size(); ++i) {
            x0 = min(x0, v[i].x);    x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y);    y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)         return pdist2(point(x0, y0), p);
            else if (p.y > y1)    return pdist2(point(x0, y1), p);
            else                  return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)         return pdist2(point(x1, y0), p);
            else if (p.y > y1)    return pdist2(point(x1, y1), p);
            else                  return pdist2(point(x1, p.y), p);
        }
        else {
            if (p.y < y0)         return pdist2(point(p.x, y0), p);
            else if (p.y > y1)    return pdist2(point(p.x, y1), p);
            else                  return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or leaf
struct kdnode
{
    bool leaf;       // true if this is a leaf node (has one point)
    point pt;        // the single point of this is a leaf
    bbox bound;      // bounding box for set of points in children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete second; }

    // intersect a point with this node (returns squared distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp)
    {
        // compute bounding box for points at this node
        bound.compute(vp);
```

```cpp
        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        }
        else {
            // split on x if the bbox is wider than high (not best heuristic...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode();    first->construct(vl);
            second = new kdnode();   second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree
{
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to nearest point
    ntype search(kdnode *node, const point &p)
    {
        if (node->leaf) {
            // commented special case tells a point not to find itself
//          if (p == node->pt) return sentry;
//          else
                return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        }
        else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};

// ----------------------------------------------------------------------------
// some basic test code here

int main()
{
    // generate some random points for a kd-tree
```

```cpp
    vector<point> vp;
    for (int i = 0; i < 100000; ++i) {
        vp.push_back(point(rand()%100000, rand()%100000));
    }
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ", " << q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }

    return 0;
}

// ----------------------------------------------------------------------------
```

## 5.5  Splay tree

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

const int N_MAX = 130010;
const int oo = 0x3f3f3f3f;
struct Node
{
    Node *ch[2], *pre;
    int val, size;
    bool isTurned;
} nodePool[N_MAX], *null, *root;

Node *allocNode(int val)
{
    static int freePos = 0;
    Node *x = &nodePool[freePos ++];
    x->val = val, x->isTurned = false;
    x->ch[0] = x->ch[1] = x->pre = null;
    x->size = 1;
    return x;
}

inline void update(Node *x)
{
    x->size = x->ch[0]->size + x->ch[1]->size + 1;
}

inline void makeTurned(Node *x)
{
    if(x == null)
        return;
    swap(x->ch[0], x->ch[1]);
    x->isTurned ^= 1;
}

inline void pushDown(Node *x)
{
    if(x->isTurned)
    {
        makeTurned(x->ch[0]);
        makeTurned(x->ch[1]);
        x->isTurned ^= 1;
    }
}

inline void rotate(Node *x, int c)
{
    Node *y = x->pre;
    x->pre = y->pre;
    if(y->pre != null)
        y->pre->ch[y == y->pre->ch[1]] = x;
    y->ch[!c] = x->ch[c];
    if(x->ch[c] != null)
        x->ch[c]->pre = y;
```

```
    x->ch[c] = y, y->pre = x;
    update(y);
    if(y == root)
        root = x;
}

void splay(Node *x, Node *p)
{
    while(x->pre != p)
    {
        if(x->pre->pre == p)
            rotate(x, x == x->pre->ch[0]);
        else
        {
            Node *y = x->pre, *z = y->pre;
            if(y == z->ch[0])
            {
                if(x == y->ch[0])
                    rotate(y, 1), rotate(x, 1);
                else
                    rotate(x, 0), rotate(x, 1);
            }
            else
            {
                if(x == y->ch[1])
                    rotate(y, 0), rotate(x, 0);
                else
                    rotate(x, 1), rotate(x, 0);
            }
        }
    }
    update(x);
}

void select(int k, Node *fa)
{
    Node *now = root;
    while(1)
    {
        pushDown(now);
        int tmp = now->ch[0]->size + 1;
        if(tmp == k)
            break;
        else if(tmp < k)
            now = now->ch[1], k -= tmp;
        else
            now = now->ch[0];
    }
    splay(now, fa);
}

Node *makeTree(Node *p, int l, int r)
{
    if(l > r)
        return null;
    int mid = (l + r) / 2;
    Node *x = allocNode(mid);
    x->pre = p;
    x->ch[0] = makeTree(x, l, mid - 1);
    x->ch[1] = makeTree(x, mid + 1, r);
    update(x);
    return x;
}

int main()
{
    int n, m;
    null = allocNode(0);
    null->size = 0;
    root = allocNode(0);
    root->ch[1] = allocNode(oo);
    root->ch[1]->pre = root;
    update(root);

    scanf("%d%d", &n, &m);
    root->ch[1]->ch[0] = makeTree(root->ch[1], 1, n);
    splay(root->ch[1]->ch[0], null);

    while(m --)
```

```
    {
        int a, b;
        scanf("%d%d", &a, &b);
        a ++, b ++;
        select(a - 1, null);
        select(b + 1, root);
        makeTurned(root->ch[1]->ch[0]);
    }

    for(int i = 1; i <= n; i ++)
    {
        select(i + 1, null);
        printf("%d ", root->val);
    }
}
```

## 5.6   Lowest common ancestor

```
const int max_nodes, log_max_nodes;
int num_nodes, log_num_nodes, root;

vector<int> children[max_nodes];        // children[i] contains the children of
                                        //    node i
int A[max_nodes][log_max_nodes+1];      // A[i][j] is the 2^j-th ancestor of
                                        //    node i, or -1 if that ancestor does not exist
int L[max_nodes];                       // L[i] is the distance between node i
                                        //    and the root

// floor of the binary logarithm of n
int lb(unsigned int n)
{
    if(n==0)
        return -1;
    int p = 0;
    if (n >= 1<<16) { n >>= 16; p += 16; }
    if (n >= 1<< 8) { n >>=  8; p +=  8; }
    if (n >= 1<< 4) { n >>=  4; p +=  4; }
    if (n >= 1<< 2) { n >>=  2; p +=  2; }
    if (n >= 1<< 1) {           p +=  1; }
    return p;
}

void DFS(int i, int l)
{
    L[i] = l;
    for(int j = 0; j < children[i].size(); j++)
        DFS(children[i][j], l+1);
}

int LCA(int p, int q)
{
    // ensure node p is at least as deep as node q
    if(L[p] < L[q])
        swap(p, q);

    // "binary search" for the ancestor of node p situated on the same level as
    //    q
    for(int i = log_num_nodes; i >= 0; i--)
        if(L[p] - (1<<i) >= L[q])
            p = A[p][i];

    if(p == q)
        return p;

    // "binary search" for the LCA
    for(int i = log_num_nodes; i >= 0; i--)
        if(A[p][i] != -1 && A[p][i] != A[q][i])
        {
            p = A[p][i];
            q = A[q][i];
        }

    return A[p][0];
}

int main(int argc,char* argv[])
```

```
{
    // read num_nodes, the total number of nodes
    log_num_nodes=lb(num_nodes);

    for(int i = 0; i < num_nodes; i++)
    {
        int p;
        // read p, the parent of node i or -1 if node i is the root

        A[i][0] = p;
        if(p != -1)
            children[p].push_back(i);
        else
            root = i;
    }

    // precompute A using dynamic programming
    for(int j = 1; j <= log_num_nodes; j++)
        for(int i = 0; i < num_nodes; i++)
            if(A[i][j-1] != -1)
                A[i][j] = A[A[i][j-1]][j-1];
            else
                A[i][j] = -1;

    // precompute L
    DFS(root, 0);


    return 0;
}
```

# 6  Miscellaneous

## 6.1  Longest increasing subsequence

```
// Given a list of numbers of length n, this routine extracts a
// longest increasing subsequence.
//
// Running time: O(n log n)
//
//   INPUT: a vector of integers
//   OUTPUT: a vector containing the longest increasing subsequence

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> VI;
typedef pair<int,int> PII;
typedef vector<PII> VPII;

#define STRICTLY_INCREASNG

VI LongestIncreasingSubsequence(VI v) {
  VPII best;
  VI dad(v.size(), -1);

  for (int i = 0; i < v.size(); i++) {
#ifdef STRICTLY_INCREASNG
    PII item = make_pair(v[i], 0);
    VPII::iterator it = lower_bound(best.begin(), best.end(), item);
    item.second = i;
#else
    PII item = make_pair(v[i], i);
    VPII::iterator it = upper_bound(best.begin(), best.end(), item);
#endif
    if (it == best.end()) {
      dad[i] = (best.size() == 0 ? -1 : best.back().second);
      best.push_back(item);
    } else {
      dad[i] = it == best.begin() ? -1 : prev(it)->second;
```

```
      *it = item;
    }
  }

  VI ret;
  for (int i = best.back().second; i >= 0; i = dad[i])
    ret.push_back(v[i]);
  reverse(ret.begin(), ret.end());
  return ret;
}
```

## 6.2  Dates

```
// Routines for performing computations on dates.  In these routines,
// months are expressed as integers from 1 to 12, days are expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.

#include <iostream>
#include <string>

using namespace std;

string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

// converts Gregorian date to integer (Julian day number)
int dateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year
void intToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;

  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string intToDay (int jd){
  return dayOfWeek[jd % 7];
}

int main (int argc, char **argv){
  int jd = dateToInt (3, 24, 2004);
  int m, d, y;
  intToDate (jd, m, d, y);
  string day = intToDay (jd);

  // expected output:
  //    2453089
  //    3/24/2004
  //    Wed
  cout << jd << endl
    << m << "/" << d << "/" << y << endl
    << day << endl;
}
```

## 6.3  Prime numbers

```cpp
// O(sqrt(x)) Exhaustive Primality Test
#include <cmath>
#define EPS 1e-7
typedef long long LL;
bool IsPrimeSlow (LL x)
{
  if(x<=1) return false;
  if(x<=3) return true;
  if (!(x%2) || !(x%3)) return false;
  LL s=(LL)(sqrt((double)(x))+EPS);
  for(LL i=5;i<=s;i+=6)
  {
    if (!(x%i) || !(x%(i+2))) return false;
  }
  return true;
}
// Primes less than 1000:
//      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    101    103    107    109    113    127    131    137    139    149    151
//    157    163    167    173    179    181    191    193    197    199    211    223
//    227    229    233    239    241    251    257    263    269    271    277    281
//    283    293    307    311    313    317    331    337    347    349    353    359
//    367    373    379    383    389    397    401    409    419    421    431    433
//    439    443    449    457    461    463    467    479    487    491    499    503
//    509    521    523    541    547    557    563    569    571    577    587    593
//    599    601    607    613    617    619    631    641    643    647    653    659
//    661    673    677    683    691    701    709    719    727    733    739    743
//    751    757    761    769    773    787    797    809    811    821    823    827
//    829    839    853    857    859    863    877    881    883    887    907    911
//    919    929    937    941    947    953    967    971    977    983    991    997

// Other primes:
//     The largest prime smaller than 10 is 7.
//     The largest prime smaller than 100 is 97.
//     The largest prime smaller than 1000 is 997.
//     The largest prime smaller than 10000 is 9973.
//     The largest prime smaller than 100000 is 99991.
//     The largest prime smaller than 1000000 is 999983.
//     The largest prime smaller than 10000000 is 9999991.
//     The largest prime smaller than 100000000 is 99999989.
//     The largest prime smaller than 1000000000 is 999999937.
//     The largest prime smaller than 10000000000 is 9999999967.
//     The largest prime smaller than 100000000000 is 99999999977.
//     The largest prime smaller than 1000000000000 is 999999999989.
//     The largest prime smaller than 10000000000000 is 9999999999971.
//     The largest prime smaller than 100000000000000 is 99999999999973.
//     The largest prime smaller than 1000000000000000 is 999999999999989.
//     The largest prime smaller than 10000000000000000 is 9999999999999937.
//     The largest prime smaller than 100000000000000000 is 99999999999999997.
//     The largest prime smaller than 1000000000000000000 is 999999999999999989.
```

## 6.4   C++ input/output

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    // Ouput a specific number of digits past the decimal point,
    // in this case 5
    cout.setf(ios::fixed); cout << setprecision(5);
    cout << 100.0/7.0 << endl;
    cout.unsetf(ios::fixed);

    // Output the decimal point and trailing zeros
    cout.setf(ios::showpoint);
    cout << 100.0 << endl;
    cout.unsetf(ios::showpoint);

    // Output a '+' before positive values
    cout.setf(ios::showpos);
    cout << 100 << " " << -100 << endl;
```

```cpp
    cout.unsetf(ios::showpos);

    // Output numerical values in hexadecimal
    cout << hex << 100 << " " << 1000 << " " << 10000 << dec << endl;
}
```

## 6.5   Knuth-Morris-Pratt

```cpp
/*
Finds all occurrences of the pattern string p within the
text string t. Running time is O(n + m), where n and m
are the lengths of p and t, respecitvely.
*/

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

using namespace std;

typedef vector<int> VI;

void buildPi(string& p, VI& pi)
{
  pi = VI(p.length());
  int k = -2;
  for(int i = 0; i < p.length(); i++) {
    while(k >= -1 && p[k+1] != p[i])
      k = (k == -1) ? -2 : pi[k];
    pi[i] = ++k;
  }
}

int KMP(string& t, string& p)
{
  VI pi;
  buildPi(p, pi);
  int k = -1;
  for(int i = 0; i < t.length(); i++) {
    while(k >= -1 && p[k+1] != t[i])
      k = (k == -1) ? -2 : pi[k];
    k++;
    if(k == p.length() - 1) {
      // p matches t[i-m+1, ..., i]
      cout << "matched at index " << i-k << ": ";
      cout << t.substr(i-k, p.length()) << endl;
      k = (k == -1) ? -2 : pi[k];
    }
  }
  return 0;
}

int main()
{
  string a = "AABAACAADAABAABA", b = "AABA";
  KMP(a, b); // expected matches at: 0, 9, 12
  return 0;
}
```

## 6.6   Latitude/longitude

```cpp
/*
Converts from rectangular coordinates to latitude/longitude and vice
versa. Uses degrees (not radians).
*/

#include <iostream>
#include <cmath>

using namespace std;

struct ll
```

```
{
  double r, lat, lon;
};

struct rect
{
  double x, y, z;
};

ll convert(rect& P)
{
  ll Q;
  Q.r = sqrt(P.x*P.x+P.y*P.y+P.z*P.z);
  Q.lat = 180/M_PI*asin(P.z/Q.r);
  Q.lon = 180/M_PI*acos(P.x/sqrt(P.x*P.x+P.y*P.y));

  return Q;
}

rect convert(ll& Q)
{
  rect P;
  P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.z = Q.r*sin(Q.lat*M_PI/180);

  return P;
}

int main()
{
  rect A;
  ll B;

  A.x = -1.0; A.y = 2.0; A.z = -3.0;

  B = convert(A);
  cout << B.r << " " << B.lat << " " << B.lon << endl;

  A = convert(B);
  cout << A.x << " " << A.y << " " << A.z << endl;
}
```

## 6.7   Miller-Rabin Primality Test (C++)

```
// Randomized Primality Test (Miller-Rabin):
//   Error rate: 2^(-TRIAL)
//   Almost constant time. srand is needed

#include <stdlib.h>
#define EPS 1e-7

typedef long long LL;

LL ModularMultiplication(LL a, LL b, LL m)
{
        LL ret=0, c=a;
        while(b)
        {
                if(b&1) ret=(ret+c)%m;
                b>>=1; c=(c+c)%m;
        }
        return ret;
}
LL ModularExponentiation(LL a, LL n, LL m)
{
        LL ret=1, c=a;
        while(n)
        {
                if(n&1) ret=ModularMultiplication(ret, c, m);
                n>>=1; c=ModularMultiplication(c, c, m);
        }
        return ret;
}
bool Witness(LL a, LL n)
```

```
{
        LL u=n-1;
  int t=0;
        while(!(u&1)){u>>=1; t++;}
        LL x0=ModularExponentiation(a, u, n), x1;
        for(int i=1;i<=t;i++)
        {
                x1=ModularMultiplication(x0, x0, n);
                if(x1==1 && x0!=1 && x0!=n-1) return true;
                x0=x1;
        }
        if(x0!=1) return true;
        return false;
}
LL Random(LL n)
{
  LL ret=rand(); ret*=32768;
        ret+=rand(); ret*=32768;
        ret+=rand(); ret*=32768;
        ret+=rand();
  return ret%n;
}
bool IsPrimeFast(LL n, int TRIAL)
{
  while(TRIAL--)
  {
    LL a=Random(n-2)+1;
    if(Witness(a, n)) return false;
  }
  return true;
}
```

## 6.8   OTHER Miller-Rabin Primality Test (C++)

```
WITNESSES : 2, 325, 9375, 28178, 450775, 9780504, 1795265022

#if !defined(_WIN64) && !defined(__amd64__)

static inline uint64_t mul128(uint64_t a, uint64_t b, uint64_t *hi)
{
        uint32_t AH = a >> 32;
        uint32_t AL = (uint32_t)a;

        uint32_t BH = b >> 32;
        uint32_t BL = (uint32_t)b;

        uint64_t AHBH = (uint64_t)AH * BH;
        uint64_t ALBL = (uint64_t)AL * BL;
        uint64_t AHBL = (uint64_t)AH * BL;
        uint64_t ALBH = (uint64_t)AL * BH;

        // take care of integer overflow

        uint64_t middle = AHBL + ALBH;

        if (middle < AHBL) AHBH += (1ULL << 32);

        uint64_t res_lo = ALBL + (middle << 32);
        if (res_lo < ALBL) AHBH++;

        AHBH += middle >> 32;

        *hi = AHBH;

        return res_lo;
}
#elif !defined(_MSC_VER)
static inline uint64_t mul128(uint64_t a, uint64_t b, uint64_t *hi)
{
        uint64_t lo;
        asm("mulq %3" : "=a"(lo), "=d"(*hi) : "a"(a), "rm"(b));
        return lo;
}
#else
```

```c
        #include <intrin.h>

        #define mul128(a, b, hi) _umul128(a, b, hi)
#endif

#if (defined(_WIN64) || defined(__amd64__)) && !defined(_MSC_VER)

#if 0
static inline uint64_t mulmod64(uint64_t a, uint64_t b, uint64_t n)
{
        return (((unsigned __int128)a) * b) % n;
}
#else
// if a*b / c > 2^64, floating point exception will occur
static inline uint64_t mulmod64(uint64_t a, uint64_t b, uint64_t n)
{
        uint64_t d;
        uint64_t unused; // dummy output, unused, to tell GCC that RAX register
                is modified by this snippet
        asm ("mulq %3\n\t"
                "divq %4"
                :"=a"(unused), "=&d"(d)
                :"a"(a), "rm"(b), "rm"(n)
                :"cc");
        return d;
}
#endif

#elif defined(_WIN64) && defined(_MSC_VER)

// according to documentation future Visual C++ versions
// may use different registers to store parameters
//
// it works at least with Visual C++ 2012 and Visual C++ 2015

// arguments passed in RCX, RDX, R8
// return value in RAX

#pragma section("mulmod64", read, execute)
__declspec(allocate("mulmod64"))
unsigned char mulmod64_code[] = {
        0x48, 0x89, 0xC8, // mov rax,rcx
        0x48, 0xF7, 0xE2, // mul rdx
        0x49, 0xF7, 0xF0, // div r8
        0x48, 0x89, 0xD0, // mov rax,rdx
        0xC3              // ret
};

uint64_t (__fastcall *mulmod64)(uint64_t, uint64_t, uint64_t) =
        (uint64_t (__fastcall *)(uint64_t, uint64_t, uint64_t))(void *)
                mulmod64_code;

#else

// requires a, b < n
static inline uint64_t mulmod64(uint64_t a, uint64_t b, uint64_t n)
{
        uint64_t r = 0;

        if (a < b) { uint64_t tmp = a; a = b; b = tmp; } // swap(a, b)

        if (n < (1ULL << 63)) {
                while (b) {
                        if (b & 1) {
                                r += a;
                                if (r >= n) r -= n;
                        }
                        b >>= 1;
                        if (b) {
                                a += a;
                                if (a >= n) a -= n;
                        }
                }
```

```c
        } else {
                while (b) {
                        if (b & 1) {
                                r = (n-r > a) ? r+a : r+a-n;
                        }
                        b >>= 1;
                        if (b) {
                                a = (n-a > a) ? a+a : a+a-n;
                        }
                }
        }
        return r;
}


static inline uint64_t modular_exponentiation64(uint64_t a, uint64_t b, uint64_t
        n)
{
        uint64_t d=1, A=a;

        do {
                if (b&1)
                        d=mulmod64(d, A, n);
                A=mulmod64(A, A, n);
        } while (b>>=1);

        return (uint64_t)d;
}

static inline int straightforward_mr64(const uint64_t bases[], int bases_cnt,
        uint64_t n)
{
        uint64_t u=n-1;

#ifdef _MSC_VER
        // u will be even, as n is required to be odd
        int t=1, j; u >>= 1;

        while (u % 2 == 0) { // while even
                t++;
                u >>= 1;
        }
#else
        int t = __builtin_ctzll(u), j;
        u >>= t;
#endif

        for (j=0; j<bases_cnt; j++) {
                uint64_t a = bases[j], x;
                int i;

                if (a >= n) a %= n;

                if (a == 0) continue;

                x = modular_exponentiation64(a, u, n);

                if (x == 1 || x == n-1) continue;

                for (i=1; i<t; i++) {
                        x=mulmod64(x, x, n);
                        if (x == 1)   return 0;
                        if (x == n-1) break;
                }

                // if we didn't break, the number is composite
                if (i == t) return 0;
        }

        return 1;
}
```

| Theoretical Computer Science Cheat Sheet | |
|---|---|
| Definitions | Series |

**Definitions**

| | |
|---|---|
| $f(n) = O(g(n))$ | iff $\exists$ positive $c, n_0$ such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$. |
| $f(n) = \Omega(g(n))$ | iff $\exists$ positive $c, n_0$ such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$. |
| $f(n) = \Theta(g(n))$ | iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. |
| $f(n) = o(g(n))$ | iff $\lim_{n \to \infty} f(n)/g(n) = 0$. |
| $\lim_{n \to \infty} a_n = a$ | iff $\forall \epsilon > 0$, $\exists n_0$ such that $|a_n - a| < \epsilon$, $\forall n \ge n_0$. |
| $\sup S$ | least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$. |
| $\inf S$ | greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$. |
| $\liminf_{n \to \infty} a_n$ | $\lim_{n \to \infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$. |
| $\limsup_{n \to \infty} a_n$ | $\lim_{n \to \infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$. |
| $\binom{n}{k}$ | Combinations: Size $k$ subsets of a size $n$ set. |
| $\left[ {n \atop k} \right]$ | Stirling numbers (1st kind): Arrangements of an $n$ element set into $k$ cycles. |
| $\left\{ {n \atop k} \right\}$ | Stirling numbers (2nd kind): Partitions of an $n$ element set into $k$ non-empty sets. |
| $\left\langle {n \atop k} \right\rangle$ | 1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \ldots \pi_n$ on $\{1, 2, \ldots, n\}$ with $k$ ascents. |
| $\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle$ | 2nd order Eulerian numbers. |
| $C_n$ | Catalan Numbers: Binary trees with $n + 1$ vertices. |

**Series**

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[ (n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left( (i+1)^{m+1} - i^{m+1} - (m+1)i^m \right) \right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m} \binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n} \frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \quad \sum_{i=1}^{n} \binom{i}{m}H_i = \binom{n+1}{m+1}\left( H_{n+1} - \frac{1}{m+1} \right).$$

**1.** $\binom{n}{k} = \dfrac{n!}{(n-k)!k!},$ **2.** $\displaystyle\sum_{k=0}^{n} \binom{n}{k} = 2^n,$ **3.** $\binom{n}{k} = \binom{n}{n-k},$

**4.** $\binom{n}{k} = \dfrac{n}{k}\binom{n-1}{k-1},$ **5.** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$

**6.** $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k},$ **7.** $\displaystyle\sum_{k=0}^{n} \binom{r+k}{k} = \binom{r+n+1}{n},$

**8.** $\displaystyle\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1},$ **9.** $\displaystyle\sum_{k=0}^{n} \binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n},$

**10.** $\binom{n}{k} = (-1)^k\binom{k-n-1}{k},$ **11.** $\left\{ {n \atop 1} \right\} = \left\{ {n \atop n} \right\} = 1,$

**12.** $\left\{ {n \atop 2} \right\} = 2^{n-1} - 1,$ **13.** $\left\{ {n \atop k} \right\} = k\left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\},$

**14.** $\left[ {n \atop 1} \right] = (n-1)!,$ **15.** $\left[ {n \atop 2} \right] = (n-1)!H_{n-1},$ **16.** $\left[ {n \atop n} \right] = 1,$ **17.** $\left[ {n \atop k} \right] \ge \left\{ {n \atop k} \right\},$

**18.** $\left[ {n \atop k} \right] = (n-1)\left[ {n-1 \atop k} \right] + \left[ {n-1 \atop k-1} \right],$ **19.** $\left\{ {n \atop n-1} \right\} = \left[ {n \atop n-1} \right] = \binom{n}{2},$ **20.** $\displaystyle\sum_{k=0}^{n} \left[ {n \atop k} \right] = n!,$ **21.** $C_n = \dfrac{1}{n+1}\binom{2n}{n},$

**22.** $\left\langle {n \atop 0} \right\rangle = \left\langle {n \atop n-1} \right\rangle = 1,$ **23.** $\left\langle {n \atop k} \right\rangle = \left\langle {n \atop n-1-k} \right\rangle,$ **24.** $\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle,$

**25.** $\left\langle {0 \atop k} \right\rangle = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\left\langle {n \atop 1} \right\rangle = 2^n - n - 1,$ **27.** $\left\langle {n \atop 2} \right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2},$

**28.** $x^n = \displaystyle\sum_{k=0}^{n} \left\langle {n \atop k} \right\rangle\binom{x+k}{n},$ **29.** $\left\langle {n \atop m} \right\rangle = \displaystyle\sum_{k=0}^{m} \binom{n+1}{k}(m+1-k)^n(-1)^k,$ **30.** $m!\left\{ {n \atop m} \right\} = \displaystyle\sum_{k=0}^{n} \left\langle {n \atop k} \right\rangle\binom{k}{n-m},$

**31.** $\left\langle {n \atop m} \right\rangle = \displaystyle\sum_{k=0}^{n} \left\{ {n \atop k} \right\}\binom{n-k}{m}(-1)^{n-k-m}k!,$ **32.** $\left\langle\!\!\left\langle {n \atop 0} \right\rangle\!\!\right\rangle = 1,$ **33.** $\left\langle\!\!\left\langle {n \atop n} \right\rangle\!\!\right\rangle = 0 \quad \text{for } n \ne 0,$

**34.** $\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle {n-1 \atop k} \right\rangle\!\!\right\rangle + (2n-1-k)\left\langle\!\!\left\langle {n-1 \atop k-1} \right\rangle\!\!\right\rangle,$ **35.** $\displaystyle\sum_{k=0}^{n} \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = \dfrac{(2n)^{\underline{n}}}{2^n},$

**36.** $\left\{ {x \atop x-n} \right\} = \displaystyle\sum_{k=0}^{n} \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle\binom{x+n-1-k}{2n},$ **37.** $\left\{ {n+1 \atop m+1} \right\} = \displaystyle\sum_{k} \binom{n}{k}\left\{ {k \atop m} \right\} = \displaystyle\sum_{k=0}^{n} \left\{ {k \atop m} \right\}(m+1)^{n-k},$

| Identities Cont. | | Trees |
|---|---|---|

**Identities Cont.**

**38.** $\left[\begin{array}{c} n+1 \\ m+1 \end{array}\right] = \sum_k \left[\begin{array}{c} n \\ k \end{array}\right]\left(\begin{array}{c} k \\ m \end{array}\right) = \sum_{k=0}^{n} \left[\begin{array}{c} k \\ m \end{array}\right] n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!}\left[\begin{array}{c} k \\ m \end{array}\right],$ **39.** $\left[\begin{array}{c} x \\ x-n \end{array}\right] = \sum_{k=0}^{n} \left\langle\!\left\langle\begin{array}{c} n \\ k \end{array}\right\rangle\!\right\rangle\left(\begin{array}{c} x+k \\ 2n \end{array}\right),$

**40.** $\left\{\begin{array}{c} n \\ m \end{array}\right\} = \sum_k \left(\begin{array}{c} n \\ k \end{array}\right)\left\{\begin{array}{c} k+1 \\ m+1 \end{array}\right\}(-1)^{n-k},$ **41.** $\left[\begin{array}{c} n \\ m \end{array}\right] = \sum_k \left[\begin{array}{c} n+1 \\ k+1 \end{array}\right]\left(\begin{array}{c} k \\ m \end{array}\right)(-1)^{m-k},$

**42.** $\left\{\begin{array}{c} m+n+1 \\ m \end{array}\right\} = \sum_{k=0}^{m} k\left\{\begin{array}{c} n+k \\ k \end{array}\right\},$ **43.** $\left[\begin{array}{c} m+n+1 \\ m \end{array}\right] = \sum_{k=0}^{m} k(n+k)\left[\begin{array}{c} n+k \\ k \end{array}\right],$

**44.** $\left(\begin{array}{c} n \\ m \end{array}\right) = \sum_k \left\{\begin{array}{c} n+1 \\ k+1 \end{array}\right\}\left[\begin{array}{c} k \\ m \end{array}\right](-1)^{m-k},$ **45.** $(n-m)!\left(\begin{array}{c} n \\ m \end{array}\right) = \sum_k \left[\begin{array}{c} n+1 \\ k+1 \end{array}\right]\left\{\begin{array}{c} k \\ m \end{array}\right\}(-1)^{m-k},$ for $n \geq m,$

**46.** $\left\{\begin{array}{c} n \\ n-m \end{array}\right\} = \sum_k \left(\begin{array}{c} m-n \\ m+k \end{array}\right)\left(\begin{array}{c} m+n \\ n+k \end{array}\right)\left[\begin{array}{c} m+k \\ k \end{array}\right],$ **47.** $\left[\begin{array}{c} n \\ n-m \end{array}\right] = \sum_k \left(\begin{array}{c} m-n \\ m+k \end{array}\right)\left(\begin{array}{c} m+n \\ n+k \end{array}\right)\left\{\begin{array}{c} m+k \\ k \end{array}\right\},$

**48.** $\left\{\begin{array}{c} n \\ \ell+m \end{array}\right\}\left(\begin{array}{c} \ell+m \\ \ell \end{array}\right) = \sum_k \left\{\begin{array}{c} k \\ \ell \end{array}\right\}\left\{\begin{array}{c} n-k \\ m \end{array}\right\}\left(\begin{array}{c} n \\ k \end{array}\right),$ **49.** $\left[\begin{array}{c} n \\ \ell+m \end{array}\right]\left(\begin{array}{c} \ell+m \\ \ell \end{array}\right) = \sum_k \left[\begin{array}{c} k \\ \ell \end{array}\right]\left[\begin{array}{c} n-k \\ m \end{array}\right]\left(\begin{array}{c} n \\ k \end{array}\right).$

**Trees**

Every tree with $n$ vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:
$$\sum_{i=1}^{n} 2^{-d_i} \leq 1,$$
and equality holds only if every internal node has 2 sons.

## Recurrences

Master method:
$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then
$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then
$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large $n$, then
$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence
$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that $T_i$ is always a power of two. Let $t_i = \log_2 T_i$. Then we have
$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by $2^{i+1}$ we get
$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find
$$u_{i+1} = \frac{1}{2} + u_i, \qquad u_1 = \frac{1}{2},$$

which is simply $u_i = i/2$. So we find that $T_i$ has the closed form $T_i = 2^{i2^{i-1}}$.

Summing factors (example): Consider the following recurrence
$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving $T$ are on the left side
$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$1\big(T(n) - 3T(n/2) = n\big)$$
$$3\big(T(n/2) - 3T(n/4) = n/2\big)$$
$$\vdots \quad \vdots \quad \vdots$$
$$3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big)$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get
$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have
$$n \sum_{i=0}^{m-1} c^i = n\left(\frac{c^m - 1}{c - 1}\right)$$
$$= 2n(c^{\log_2 n} - 1)$$
$$= 2n(c^{(k-1)\log_c n} - 1)$$
$$= 2n^k - 2n,$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider
$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that
$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find
$$T_{i+1} - T_i = 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j$$
$$= T_i.$$

And so $T_{i+1} = 2T_i = 2^{i+1}.$

Generating functions:
1. Multiply both sides of the equation by $x^i$.
2. Sum both sides over all $i$ for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of $x^i$ in $G(x)$ is $g_i$.

Example:
$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:
$$\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$$

We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:
$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:
$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:
$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:
$$G(x) = x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right)$$
$$= x\left(2\sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i\right)$$
$$= \sum_{i \geq 0}(2^{i+1} - 1)x^{i+1}.$$

So $g_i = 2^i - 1.$

$\pi \approx 3.14159,$ $\quad e \approx 2.71828,$ $\quad \gamma \approx 0.57721,$ $\quad \phi = \frac{1+\sqrt{5}}{2} \approx 1.61803,$ $\quad \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

| $i$ | $2^i$ | $p_i$ |
|---|---|---|
| 1 | 2 | 2 |
| 2 | 4 | 3 |
| 3 | 8 | 5 |
| 4 | 16 | 7 |
| 5 | 32 | 11 |
| 6 | 64 | 13 |
| 7 | 128 | 17 |
| 8 | 256 | 19 |
| 9 | 512 | 23 |
| 10 | 1,024 | 29 |
| 11 | 2,048 | 31 |
| 12 | 4,096 | 37 |
| 13 | 8,192 | 41 |
| 14 | 16,384 | 43 |
| 15 | 32,768 | 47 |
| 16 | 65,536 | 53 |
| 17 | 131,072 | 59 |
| 18 | 262,144 | 61 |
| 19 | 524,288 | 67 |
| 20 | 1,048,576 | 71 |
| 21 | 2,097,152 | 73 |
| 22 | 4,194,304 | 79 |
| 23 | 8,388,608 | 83 |
| 24 | 16,777,216 | 89 |
| 25 | 33,554,432 | 97 |
| 26 | 67,108,864 | 101 |
| 27 | 134,217,728 | 103 |
| 28 | 268,435,456 | 107 |
| 29 | 536,870,912 | 109 |
| 30 | 1,073,741,824 | 113 |
| 31 | 2,147,483,648 | 127 |
| 32 | 4,294,967,296 | 131 |

**Pascal's Triangle**

```
            1
           1 1
          1 2 1
         1 3 3 1
        1 4 6 4 1
       1 5 10 10 5 1
      1 6 15 20 15 6 1
     1 7 21 35 35 21 7 1
    1 8 28 56 70 56 28 8 1
   1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

### General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):
$$B_0 = 1, \ B_1 = -\tfrac{1}{2}, \ B_2 = \tfrac{1}{6}, \ B_4 = -\tfrac{1}{30},$$
$$B_6 = \tfrac{1}{42}, \ B_8 = -\tfrac{1}{30}, \ B_{10} = \tfrac{5}{66}.$$

Change of base, quadratic formula:
$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number $e$:
$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$
$$\lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$
$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$
$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:
$$1, \tfrac{3}{2}, \tfrac{11}{6}, \tfrac{25}{12}, \tfrac{137}{60}, \tfrac{49}{20}, \tfrac{363}{140}, \tfrac{761}{280}, \tfrac{7129}{2520}, \ldots$$
$$\ln n < H_n < \ln n + 1,$$
$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:
$$1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \ldots$$
$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:
$$a(i, j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$$
$$\alpha(i) = \min\{j \mid a(j, j) \geq i\}.$$

Binomial distribution:
$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$
$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:
$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:
$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are $n$ different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all $n$ types is
$$nH_n.$$

### Probability

Continuous distributions: If
$$\Pr[a < X < b] = \int_a^b p(x)\,dx,$$
then $p$ is the probability density function of $X$. If
$$\Pr[X < a] = P(a),$$
then $P$ is the distribution function of $X$. If $P$ and $p$ both exist then
$$P(a) = \int_{-\infty}^a p(x)\,dx.$$

Expectation: If $X$ is discrete
$$\mathrm{E}[g(X)] = \sum_x g(x)\Pr[X = x].$$
If $X$ continuous then
$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx = \int_{-\infty}^{\infty} g(x)\,dP(x).$$

Variance, standard deviation:
$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$
$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events $A$ and $B$:
$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$
$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$
$$\text{iff } A \text{ and } B \text{ are independent.}$$
$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables $X$ and $Y$:
$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X] \cdot \mathrm{E}[Y],$$
$$\text{if } X \text{ and } Y \text{ are independent.}$$
$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$
$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:
$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n}\Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:
$$\Pr\left[\bigvee_{i=1}^{n} X_i\right] = \sum_{i=1}^{n}\Pr[X_i] +$$
$$\sum_{k=2}^{n}(-1)^{k+1}\sum_{i_i < \cdots < i_k}\Pr\left[\bigwedge_{j=1}^{k} X_{i_j}\right].$$

Moment inequalities:
$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$
$$\Pr\left[\left|X - \mathrm{E}[X]\right| \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.$$

Geometric distribution:
$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$
$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$

# Theoretical Computer Science Cheat Sheet

## Trigonometry
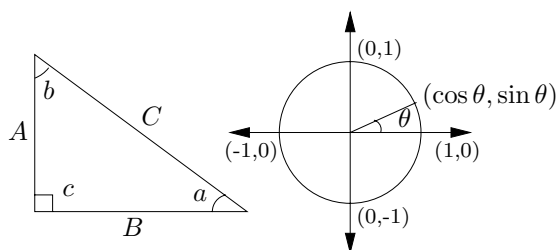


Pythagorean theorem:
$$C^2 = A^2 + B^2.$$

Definitions:
$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:
$$\tfrac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:
$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot \tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2\sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x+y)\sin(x-y) = \sin^2 x - \sin^2 y,$$
$$\cos(x+y)\cos(x-y) = \cos^2 x - \sin^2 y.$$

Euler's equation:
$$e^{ix} = \cos x + i\sin x, \qquad e^{i\pi} = -1.$$

## Matrices

Multiplication:
$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff $A$ is non-singular.
$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

$2 \times 2$ and $3 \times 3$ determinant:
$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$
$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g\begin{vmatrix} b & c \\ e & f \end{vmatrix} - h\begin{vmatrix} a & c \\ d & f \end{vmatrix} + i\begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= \begin{array}{l} aei + bfg + cdh \\ - ceg - fha - ibd. \end{array}$$

Permanents:
$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

## Hyperbolic Functions

Definitions:
$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

Identities:
$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2\sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
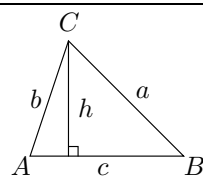$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

| $\theta$ | $\sin\theta$ | $\cos\theta$ | $\tan\theta$ |
|---|---|---|---|
| $0$ | $0$ | $1$ | $0$ |
| $\frac{\pi}{6}$ | $\frac{1}{2}$ | $\frac{\sqrt{3}}{2}$ | $\frac{\sqrt{3}}{3}$ |
| $\frac{\pi}{4}$ | $\frac{\sqrt{2}}{2}$ | $\frac{\sqrt{2}}{2}$ | $1$ |
| $\frac{\pi}{3}$ | $\frac{\sqrt{3}}{2}$ | $\frac{1}{2}$ | $\sqrt{3}$ |
| $\frac{\pi}{2}$ | $1$ | $0$ | $\infty$ |

... in mathematics you don't understand things, you just get used to them.
– J. von Neumann

## More Trig.



Law of cosines:
$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:
$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2\sin C}.$$

Heron's formula:
$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a + b + c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:
$$\sin \tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos \tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan \tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot \tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$

# Theoretical Computer Science Cheat Sheet

## Number Theory

The Chinese remainder theorem: There exists a number $C$ such that:

$$C \equiv r_1 \bmod m_1$$
$$\vdots \quad \vdots \quad \vdots$$
$$C \equiv r_n \bmod m_n$$

if $m_i$ and $m_j$ are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than $x$ relatively prime to $x$. If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of $x$ then

$$\phi(x) = \prod_{i=1}^{n} p_i^{e_i-1}(p_i - 1).$$

Euler's theorem: If $a$ and $b$ are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:
$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of $x$ then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^{n} \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: $x$ is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: $n$ is a prime iff

$$(n - 1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\left(\frac{a}{d}\right).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n}$$
$$+ O\left(\frac{n}{\ln n}\right),$$
$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3}$$
$$+ O\left(\frac{n}{(\ln n)^4}\right).$$

## Graph Theory

Definitions:

| | |
|---|---|
| *Loop* | An edge connecting a vertex to itself. |
| *Directed* | Each edge has a direction. |
| *Simple* | Graph with no loops or multi-edges. |
| *Walk* | A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$. |
| *Trail* | A walk with distinct edges. |
| *Path* | A trail with distinct vertices. |
| *Connected* | A graph where there exists a path between any two vertices. |
| *Component* | A maximal connected subgraph. |
| *Tree* | A connected acyclic graph. |
| *Free tree* | A tree with no root. |
| *DAG* | Directed acyclic graph. |
| *Eulerian* | Graph with a trail visiting each edge exactly once. |
| *Hamiltonian* | Graph with a cycle visiting each vertex exactly once. |
| *Cut* | A set of edges whose removal increases the number of components. |
| *Cut-set* | A minimal cut. |
| *Cut edge* | A size 1 cut. |
| *k-Connected* | A graph connected with the removal of any $k - 1$ vertices. |
| *k-Tough* | $\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \lvert S \rvert$. |
| *k-Regular* | A graph where all vertices have degree $k$. |
| *k-Factor* | A $k$-regular spanning subgraph. |
| *Matching* | A set of edges, no two of which are adjacent. |
| *Clique* | A set of vertices, all of which are adjacent. |
| *Ind. set* | A set of vertices, none of which are adjacent. |
| *Vertex cover* | A set of vertices which cover all edges. |
| *Planar graph* | A graph which can be embeded in the plane. |
| *Plane graph* | An embedding of a planar graph. |

$$\sum_{v \in V} \deg(v) = 2m.$$

If $G$ is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree $\leq 5$.

Notation:

| | |
|---|---|
| $E(G)$ | Edge set |
| $V(G)$ | Vertex set |
| $c(G)$ | Number of components |
| $G[S]$ | Induced subgraph |
| $\deg(v)$ | Degree of $v$ |
| $\Delta(G)$ | Maximum degree |
| $\delta(G)$ | Minimum degree |
| $\chi(G)$ | Chromatic number |
| $\chi_E(G)$ | Edge chromatic number |
| $G^c$ | Complement graph |
| $K_n$ | Complete graph |
| $K_{n_1,n_2}$ | Complete bipartite graph |
| $r(k, \ell)$ | Ramsey number |

## Geometry

Projective coordinates: triples $(x, y, z)$, not all $x$, $y$ and $z$ zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

| Cartesian | Projective |
|---|---|
| $(x, y)$ | $(x, y, 1)$ |
| $y = mx + b$ | $(m, -1, b)$ |
| $x = c$ | $(1, 0, -c)$ |

Distance formula, $L_p$ and $L_\infty$ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$
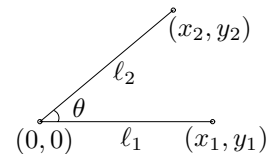$$\left[ \lvert x_1 - x_0 \rvert^p + \lvert y_1 - y_0 \rvert^p \right]^{1/p},$$
$$\lim_{p \to \infty} \left[ \lvert x_1 - x_0 \rvert^p + \lvert y_1 - y_0 \rvert^p \right]^{1/p}.$$

Area of triangle $(x_0, y_0)$, $(x_1, y_1)$ and $(x_2, y_2)$:

$$\tfrac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points $(x_0, y_0)$ and $(x_1, y_1)$:

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

| $\pi$ | Calculus |
|---|---|

## $\pi$

Wallis' identity:
$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:
$$\frac{\pi}{4} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cdots}}}}$$

Gregrory's series:
$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Newton's series:
$$\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:
$$\frac{\pi}{6} = \frac{1}{\sqrt{3}}\left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots\right)$$

Euler's series:
$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$$
$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$$
$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$$

## Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of $x$. We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of $N$ is greater than or equal to the degree of $D$, divide $N$ by $D$, obtaining
$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of $N'$ is less than that of $D$. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:
$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where
$$A = \left[\frac{N(x)}{D(x)}\right]_{x=a}.$$

For a repeated factor:
$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where
$$A_k = \frac{1}{k!}\left[\frac{d^k}{dx^k}\left(\frac{N(x)}{D(x)}\right)\right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
– George Bernard Shaw

## Calculus

Derivatives:

**1.** $\dfrac{d(cu)}{dx} = c\dfrac{du}{dx},$  **2.** $\dfrac{d(u+v)}{dx} = \dfrac{du}{dx} + \dfrac{dv}{dx},$  **3.** $\dfrac{d(uv)}{dx} = u\dfrac{dv}{dx} + v\dfrac{du}{dx},$

**4.** $\dfrac{d(u^n)}{dx} = nu^{n-1}\dfrac{du}{dx},$  **5.** $\dfrac{d(u/v)}{dx} = \dfrac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2},$  **6.** $\dfrac{d(e^{cu})}{dx} = ce^{cu}\dfrac{du}{dx},$

**7.** $\dfrac{d(c^u)}{dx} = (\ln c)c^u\dfrac{du}{dx},$  **8.** $\dfrac{d(\ln u)}{dx} = \dfrac{1}{u}\dfrac{du}{dx},$

**9.** $\dfrac{d(\sin u)}{dx} = \cos u\dfrac{du}{dx},$  **10.** $\dfrac{d(\cos u)}{dx} = -\sin u\dfrac{du}{dx},$

**11.** $\dfrac{d(\tan u)}{dx} = \sec^2 u\dfrac{du}{dx},$  **12.** $\dfrac{d(\cot u)}{dx} = \csc^2 u\dfrac{du}{dx},$

**13.** $\dfrac{d(\sec u)}{dx} = \tan u \sec u\dfrac{du}{dx},$  **14.** $\dfrac{d(\csc u)}{dx} = -\cot u \csc u\dfrac{du}{dx},$

**15.** $\dfrac{d(\arcsin u)}{dx} = \dfrac{1}{\sqrt{1-u^2}}\dfrac{du}{dx},$  **16.** $\dfrac{d(\arccos u)}{dx} = \dfrac{-1}{\sqrt{1-u^2}}\dfrac{du}{dx},$

**17.** $\dfrac{d(\arctan u)}{dx} = \dfrac{1}{1+u^2}\dfrac{du}{dx},$  **18.** $\dfrac{d(\text{arccot}\, u)}{dx} = \dfrac{-1}{1+u^2}\dfrac{du}{dx},$

**19.** $\dfrac{d(\text{arcsec}\, u)}{dx} = \dfrac{1}{u\sqrt{1-u^2}}\dfrac{du}{dx},$  **20.** $\dfrac{d(\text{arccsc}\, u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx},$

**21.** $\dfrac{d(\sinh u)}{dx} = \cosh u\dfrac{du}{dx},$  **22.** $\dfrac{d(\cosh u)}{dx} = \sinh u\dfrac{du}{dx},$

**23.** $\dfrac{d(\tanh u)}{dx} = \text{sech}^2 u\dfrac{du}{dx},$  **24.** $\dfrac{d(\coth u)}{dx} = -\text{csch}^2 u\dfrac{du}{dx},$

**25.** $\dfrac{d(\text{sech}\, u)}{dx} = -\text{sech}\, u \tanh u\dfrac{du}{dx},$  **26.** $\dfrac{d(\text{csch}\, u)}{dx} = -\text{csch}\, u \coth u\dfrac{du}{dx},$

**27.** $\dfrac{d(\text{arcsinh}\, u)}{dx} = \dfrac{1}{\sqrt{1+u^2}}\dfrac{du}{dx},$  **28.** $\dfrac{d(\text{arccosh}\, u)}{dx} = \dfrac{1}{\sqrt{u^2-1}}\dfrac{du}{dx},$

**29.** $\dfrac{d(\text{arctanh}\, u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx},$  **30.** $\dfrac{d(\text{arccoth}\, u)}{dx} = \dfrac{1}{u^2-1}\dfrac{du}{dx},$

**31.** $\dfrac{d(\text{arcsech}\, u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx},$  **32.** $\dfrac{d(\text{arccsch}\, u)}{dx} = \dfrac{-1}{|u|\sqrt{1+u^2}}\dfrac{du}{dx}.$

Integrals:

**1.** $\displaystyle\int cu\,dx = c\int u\,dx,$  **2.** $\displaystyle\int (u+v)\,dx = \int u\,dx + \int v\,dx,$

**3.** $\displaystyle\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1,$  **4.** $\displaystyle\int \frac{1}{x}dx = \ln x,$  **5.** $\displaystyle\int e^x\,dx = e^x,$

**6.** $\displaystyle\int \frac{dx}{1+x^2} = \arctan x,$  **7.** $\displaystyle\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx,$

**8.** $\displaystyle\int \sin x\,dx = -\cos x,$  **9.** $\displaystyle\int \cos x\,dx = \sin x,$

**10.** $\displaystyle\int \tan x\,dx = -\ln|\cos x|,$  **11.** $\displaystyle\int \cot x\,dx = \ln|\cos x|,$

**12.** $\displaystyle\int \sec x\,dx = \ln|\sec x + \tan x|,$  **13.** $\displaystyle\int \csc x\,dx = \ln|\csc x + \cot x|,$

**14.** $\displaystyle\int \arcsin \frac{x}{a}dx = \arcsin \frac{x}{a} + \sqrt{a^2 - x^2}, \quad a > 0,$

Calculus Cont.

**15.** $\int \arccos \frac{x}{a} dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$

**16.** $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2), \quad a > 0,$

**17.** $\int \sin^2(ax) dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

**18.** $\int \cos^2(ax) dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

**19.** $\int \sec^2 x\, dx = \tan x,$

**20.** $\int \csc^2 x\, dx = -\cot x,$

**21.** $\int \sin^n x\, dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x\, dx,$

**22.** $\int \cos^n x\, dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x\, dx,$

**23.** $\int \tan^n x\, dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x\, dx, \quad n \neq 1,$

**24.** $\int \cot^n x\, dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x\, dx, \quad n \neq 1,$

**25.** $\int \sec^n x\, dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x\, dx, \quad n \neq 1,$

**26.** $\int \csc^n x\, dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x\, dx, \quad n \neq 1,$ **27.** $\int \sinh x\, dx = \cosh x,$ **28.** $\int \cosh x\, dx = \sinh x,$

**29.** $\int \tanh x\, dx = \ln |\cosh x|,$ **30.** $\int \coth x\, dx = \ln |\sinh x|,$ **31.** $\int \operatorname{sech} x\, dx = \arctan \sinh x,$ **32.** $\int \operatorname{csch} x\, dx = \ln \left|\tanh \frac{x}{2}\right|,$

**33.** $\int \sinh^2 x\, dx = \frac{1}{4}\sinh(2x) - \frac{1}{2}x,$

**34.** $\int \cosh^2 x\, dx = \frac{1}{4}\sinh(2x) + \frac{1}{2}x,$

**35.** $\int \operatorname{sech}^2 x\, dx = \tanh x,$

**36.** $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$

**37.** $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$

**38.** $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \dfrac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \dfrac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$

**39.** $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln\left(x + \sqrt{a^2 + x^2}\right), \quad a > 0,$

**40.** $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$

**41.** $\int \sqrt{a^2 - x^2}\, dx = \frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$

**42.** $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8}(5a^2 - 2x^2)\sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$

**43.** $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$ **44.** $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left|\frac{a+x}{a-x}\right|,$ **45.** $\int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2 - x^2}},$

**46.** $\int \sqrt{a^2 \pm x^2}\, dx = \frac{x}{2}\sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln \left|x + \sqrt{a^2 \pm x^2}\right|,$

**47.** $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left|x + \sqrt{x^2 - a^2}\right|, \quad a > 0,$

**48.** $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left|\frac{x}{a + bx}\right|,$

**49.** $\int x\sqrt{a + bx}\, dx = \frac{2(3bx - 2a)(a + bx)^{3/2}}{15b^2},$

**50.** $\int \frac{\sqrt{a + bx}}{x} dx = 2\sqrt{a + bx} + a \int \frac{1}{x\sqrt{a + bx}} dx,$

**51.** $\int \frac{x}{\sqrt{a + bx}} dx = \frac{1}{\sqrt{2}} \ln \left|\frac{\sqrt{a + bx} - \sqrt{a}}{\sqrt{a + bx} + \sqrt{a}}\right|, \quad a > 0,$

**52.** $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$

**53.** $\int x\sqrt{a^2 - x^2}\, dx = -\frac{1}{3}(a^2 - x^2)^{3/2},$

**54.** $\int x^2 \sqrt{a^2 - x^2}\, dx = \frac{x}{8}(2x^2 - a^2)\sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$

**55.** $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$

**56.** $\int \frac{x\, dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$

**57.** $\int \frac{x^2\, dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$

**58.** $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left|\frac{a + \sqrt{a^2 + x^2}}{x}\right|,$

**59.** $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$

**60.** $\int x\sqrt{x^2 \pm a^2}\, dx = \frac{1}{3}(x^2 \pm a^2)^{3/2},$

**61.** $\int \frac{dx}{x\sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left|\frac{x}{a + \sqrt{a^2 + x^2}}\right|,$

| Calculus Cont. | Finite Calculus |
|---|---|

## Calculus Cont.

**62.** $\int \dfrac{dx}{x\sqrt{x^2 - a^2}} = \frac{1}{a}\arccos\frac{a}{|x|}, \quad a > 0,$ **63.** $\int \dfrac{dx}{x^2\sqrt{x^2 \pm a^2}} = \mp\dfrac{\sqrt{x^2 \pm a^2}}{a^2 x},$

**64.** $\int \dfrac{x\,dx}{\sqrt{x^2 \pm a^2}} = \sqrt{x^2 \pm a^2},$ **65.** $\int \dfrac{\sqrt{x^2 \pm a^2}}{x^4}\,dx = \mp\dfrac{(x^2 + a^2)^{3/2}}{3a^2 x^3},$

**66.** $\int \dfrac{dx}{ax^2 + bx + c} = \begin{cases} \dfrac{1}{\sqrt{b^2 - 4ac}}\ln\left|\dfrac{2ax + b - \sqrt{b^2 - 4ac}}{2ax + b + \sqrt{b^2 - 4ac}}\right|, & \text{if } b^2 > 4ac, \\[3mm] \dfrac{2}{\sqrt{4ac - b^2}}\arctan\dfrac{2ax + b}{\sqrt{4ac - b^2}}, & \text{if } b^2 < 4ac, \end{cases}$

**67.** $\int \dfrac{dx}{\sqrt{ax^2 + bx + c}} = \begin{cases} \dfrac{1}{\sqrt{a}}\ln\left|2ax + b + 2\sqrt{a}\sqrt{ax^2 + bx + c}\right|, & \text{if } a > 0, \\[3mm] \dfrac{1}{\sqrt{-a}}\arcsin\dfrac{-2ax - b}{\sqrt{b^2 - 4ac}}, & \text{if } a < 0, \end{cases}$

**68.** $\int \sqrt{ax^2 + bx + c}\,dx = \dfrac{2ax + b}{4a}\sqrt{ax^2 + bx + c} + \dfrac{4ax - b^2}{8a}\int\dfrac{dx}{\sqrt{ax^2 + bx + c}},$

**69.** $\int \dfrac{x\,dx}{\sqrt{ax^2 + bx + c}} = \dfrac{\sqrt{ax^2 + bx + c}}{a} - \dfrac{b}{2a}\int\dfrac{dx}{\sqrt{ax^2 + bx + c}},$

**70.** $\int \dfrac{dx}{x\sqrt{ax^2 + bx + c}} = \begin{cases} \dfrac{-1}{\sqrt{c}}\ln\left|\dfrac{2\sqrt{c}\sqrt{ax^2 + bx + c} + bx + 2c}{x}\right|, & \text{if } c > 0, \\[3mm] \dfrac{1}{\sqrt{-c}}\arcsin\dfrac{bx + 2c}{|x|\sqrt{b^2 - 4ac}}, & \text{if } c < 0, \end{cases}$

**71.** $\int x^3\sqrt{x^2 + a^2}\,dx = (\frac{1}{3}x^2 - \frac{2}{15}a^2)(x^2 + a^2)^{3/2},$

**72.** $\int x^n\sin(ax)\,dx = -\frac{1}{a}x^n\cos(ax) + \frac{n}{a}\int x^{n-1}\cos(ax)\,dx,$

**73.** $\int x^n\cos(ax)\,dx = \frac{1}{a}x^n\sin(ax) - \frac{n}{a}\int x^{n-1}\sin(ax)\,dx,$

**74.** $\int x^n e^{ax}\,dx = \dfrac{x^n e^{ax}}{a} - \dfrac{n}{a}\int x^{n-1}e^{ax}\,dx,$

**75.** $\int x^n\ln(ax)\,dx = x^{n+1}\left(\dfrac{\ln(ax)}{n + 1} - \dfrac{1}{(n+1)^2}\right),$

**76.** $\int x^n(\ln ax)^m\,dx = \dfrac{x^{n+1}}{n + 1}(\ln ax)^m - \dfrac{m}{n + 1}\int x^n(\ln ax)^{m-1}\,dx.$

| | | | |
|---|---|---|---|
| $x^1 =$ | $x^{\underline{1}}$ | $=$ | $x^{\overline{1}}$ |
| $x^2 =$ | $x^{\underline{2}} + x^{\underline{1}}$ | $=$ | $x^{\overline{2}} - x^{\overline{1}}$ |
| $x^3 =$ | $x^{\underline{3}} + 3x^{\underline{2}} + x^{\underline{1}}$ | $=$ | $x^{\overline{3}} - 3x^{\overline{2}} + x^{\overline{1}}$ |
| $x^4 =$ | $x^{\underline{4}} + 6x^{\underline{3}} + 7x^{\underline{2}} + x^{\underline{1}}$ | $=$ | $x^{\overline{4}} - 6x^{\overline{3}} + 7x^{\overline{2}} - x^{\overline{1}}$ |
| $x^5 =$ | $x^{\underline{5}} + 15x^{\underline{4}} + 25x^{\underline{3}} + 10x^{\underline{2}} + x^{\underline{1}}$ | $=$ | $x^{\overline{5}} - 15x^{\overline{4}} + 25x^{\overline{3}} - 10x^{\overline{2}} + x^{\overline{1}}$ |

| | | | |
|---|---|---|---|
| $x^{\overline{1}} =$ | $x^1$ | $x^{\underline{1}} =$ | $x^1$ |
| $x^{\overline{2}} =$ | $x^2 + x^1$ | $x^{\underline{2}} =$ | $x^2 - x^1$ |
| $x^{\overline{3}} =$ | $x^3 + 3x^2 + 2x^1$ | $x^{\underline{3}} =$ | $x^3 - 3x^2 + 2x^1$ |
| $x^{\overline{4}} =$ | $x^4 + 6x^3 + 11x^2 + 6x^1$ | $x^{\underline{4}} =$ | $x^4 - 6x^3 + 11x^2 - 6x^1$ |
| $x^{\overline{5}} =$ | $x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1$ | $x^{\underline{5}} =$ | $x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1$ |

## Finite Calculus

Difference, shift operators:
$$\Delta f(x) = f(x + 1) - f(x),$$
$$\mathrm{E}\,f(x) = f(x + 1).$$

Fundamental Theorem:
$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x) + C.$$
$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:
$$\Delta(cu) = c\Delta u, \qquad \Delta(u + v) = \Delta u + \Delta v,$$
$$\Delta(uv) = u\Delta v + \mathrm{E}\,v\Delta u,$$
$$\Delta(x^{\underline{n}}) = nx^{\underline{n-1}},$$
$$\Delta(H_x) = x^{\underline{-1}}, \qquad\qquad \Delta(2^x) = 2^x,$$
$$\Delta(c^x) = (c - 1)c^x, \qquad \Delta\binom{x}{m} = \binom{x}{m-1}.$$

Sums:
$$\sum cu\,\delta x = c\sum u\,\delta x,$$
$$\sum(u + v)\,\delta x = \sum u\,\delta x + \sum v\,\delta x,$$
$$\sum u\Delta v\,\delta x = uv - \sum \mathrm{E}\,v\Delta u\,\delta x,$$
$$\sum x^{\underline{n}}\delta x = \frac{x^{\underline{n+1}}}{m+1}, \qquad \sum x^{\underline{-1}}\delta x = H_x,$$
$$\sum c^x\,\delta x = \frac{c^x}{c-1}, \qquad \sum\binom{x}{m}\delta x = \binom{x}{m+1}.$$

Falling Factorial Powers:
$$x^{\underline{n}} = x(x - 1)\cdots(x - n + 1), \quad n > 0,$$
$$x^{\underline{0}} = 1,$$
$$x^{\underline{n}} = \frac{1}{(x + 1)\cdots(x + |n|)}, \quad n < 0,$$
$$x^{\underline{n+m}} = x^{\underline{m}}(x - m)^{\underline{n}}.$$

Rising Factorial Powers:
$$x^{\overline{n}} = x(x + 1)\cdots(x + n - 1), \quad n > 0,$$
$$x^{\overline{0}} = 1,$$
$$x^{\overline{n}} = \frac{1}{(x - 1)\cdots(x - |n|)}, \quad n < 0,$$
$$x^{\overline{n+m}} = x^{\overline{m}}(x + m)^{\overline{n}}.$$

Conversion:
$$x^{\underline{n}} = (-1)^n(-x)^{\overline{n}} = (x - n + 1)^{\overline{n}}$$
$$= 1/(x + 1)^{\overline{-n}},$$
$$x^{\overline{n}} = (-1)^n(-x)^{\underline{n}} = (x + n - 1)^{\underline{n}}$$
$$= 1/(x - 1)^{\underline{-n}},$$
$$x^n = \sum_{k=1}^n \left\{{n \atop k}\right\}x^{\underline{k}} = \sum_{k=1}^n \left\{{n \atop k}\right\}(-1)^{n-k}x^{\overline{k}},$$
$$x^{\underline{n}} = \sum_{k=1}^n \left[{n \atop k}\right](-1)^{n-k}x^k,$$
$$x^{\overline{n}} = \sum_{k=1}^n \left[{n \atop k}\right]x^k.$$

## Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{i=0}^{\infty} x^i,$$

$$\frac{1}{1-cx} = 1 + cx + c^2x^2 + c^3x^3 + \cdots = \sum_{i=0}^{\infty} c^i x^i,$$

$$\frac{1}{1-x^n} = 1 + x^n + x^{2n} + x^{3n} + \cdots = \sum_{i=0}^{\infty} x^{ni},$$

$$\frac{x}{(1-x)^2} = x + 2x^2 + 3x^3 + 4x^4 + \cdots = \sum_{i=0}^{\infty} i x^i,$$

$$x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) = x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots = \sum_{i=0}^{\infty} i^n x^i,$$

$$e^x = 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

$$\ln(1+x) = x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots = \sum_{i=1}^{\infty} (-1)^{i+1}\frac{x^i}{i},$$

$$\ln\frac{1}{1-x} = x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots = \sum_{i=1}^{\infty} \frac{x^i}{i},$$

$$\sin x = x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$$

$$\cos x = 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},$$

$$\tan^{-1} x = x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},$$

$$(1+x)^n = 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{n}{i} x^i,$$

$$\frac{1}{(1-x)^{n+1}} = 1 + (n+1)x + \binom{n+2}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,$$

$$\frac{x}{e^x - 1} = 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},$$

$$\frac{1}{2x}(1 - \sqrt{1-4x}) = 1 + x + 2x^2 + 5x^3 + \cdots = \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}} = 1 + x + 2x^2 + 6x^3 + \cdots = \sum_{i=0}^{\infty} \binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = 1 + (2+n)x + \binom{4+n}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,$$

$$\frac{1}{1-x}\ln\frac{1}{1-x} = x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots = \sum_{i=1}^{\infty} H_i x^i,$$

$$\frac{1}{2}\left(\ln\frac{1}{1-x}\right)^2 = \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots = \sum_{i=2}^{\infty} \frac{H_{i-1}x^i}{i},$$

$$\frac{x}{1-x-x^2} = x + x^2 + 2x^3 + 3x^4 + \cdots = \sum_{i=0}^{\infty} F_i x^i,$$

$$\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} = F_n x + F_{2n}x^2 + F_{3n}x^3 + \cdots = \sum_{i=0}^{\infty} F_{ni} x^i.$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y)\sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$x A'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x)\,dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty}\left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker

# Theoretical Computer Science Cheat Sheet

## Series

Expansions:

$$\frac{1}{(1-x)^{n+1}}\ln\frac{1}{1-x} = \sum_{i=0}^{\infty}(H_{n+i}-H_n)\binom{n+i}{i}x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty}\begin{bmatrix}n\\i\end{bmatrix}x^i,$$

$$\left(\ln\frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty}\begin{bmatrix}i\\n\end{bmatrix}\frac{n!x^i}{i!},$$

$$\tan x = \sum_{i=1}^{\infty}(-1)^{i-1}\frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\mu(i)}{i^x},$$

$$\zeta(x) = \prod_{p}\frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty}\frac{d(i)}{x^i} \quad \text{where } d(n)=\sum_{d|n}1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty}\frac{S(i)}{x^i} \quad \text{where } S(n)=\sum_{d|n}d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!}\pi^{2n}, \quad n\in\mathbb{N},$$

$$\frac{x}{\sin x} = \sum_{i=0}^{\infty}(-1)^{i-1}\frac{(4^i-2)B_{2i}x^{2i}}{(2i)!},$$

$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty}\frac{n(2i+n-1)!}{i!(n+i)!}x^i,$$

$$e^x\sin x = \sum_{i=1}^{\infty}\frac{2^{i/2}\sin\frac{i\pi}{4}}{i!}x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty}\frac{(4i)!}{16^i\sqrt{2}(2i)!(2i+1)!}x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty}\frac{4^i i!^2}{(i+1)(2i+1)!}x^{2i}.$$

$$\left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty}\begin{Bmatrix}i\\n\end{Bmatrix}x^i,$$

$$(e^x-1)^n = \sum_{i=0}^{\infty}\begin{Bmatrix}i\\n\end{Bmatrix}\frac{n!x^i}{i!},$$

$$x\cot x = \sum_{i=0}^{\infty}\frac{(-4)^i B_{2i}x^{2i}}{(2i)!},$$

$$\zeta(x) = \sum_{i=1}^{\infty}\frac{1}{i^x},$$

$$\frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\phi(i)}{i^x},$$

## Escher's Knot
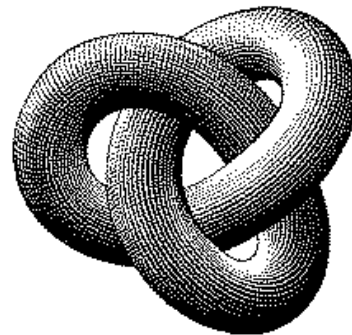


## Stieltjes Integration

If $G$ is continuous in the interval $[a,b]$ and $F$ is nondecreasing then

$$\int_a^b G(x)\,dF(x)$$

exists. If $a\le b\le c$ then

$$\int_a^c G(x)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_b^c G(x)\,dF(x).$$

If the integrals involved exist

$$\int_a^b\big(G(x)+H(x)\big)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_a^b H(x)\,dF(x),$$

$$\int_a^b G(x)\,d\big(F(x)+H(x)\big) = \int_a^b G(x)\,dF(x) + \int_a^b G(x)\,dH(x),$$

$$\int_a^b c\cdot G(x)\,dF(x) = \int_a^b G(x)\,d\big(c\cdot F(x)\big) = c\int_a^b G(x)\,dF(x),$$

$$\int_a^b G(x)\,dF(x) = G(b)F(b)-G(a)F(a)-\int_a^b F(x)\,dG(x).$$

If the integrals involved exist, and $F$ possesses a derivative $F'$ at every point in $[a,b]$ then

$$\int_a^b G(x)\,dF(x) = \int_a^b G(x)F'(x)\,dx.$$

## Cramer's Rule

If we have equations:

$$a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1$$
$$a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2$$
$$\vdots \qquad \vdots \qquad\qquad \vdots$$
$$a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$$

Let $A=(a_{i,j})$ and $B$ be the column matrix $(b_i)$. Then there is a unique solution iff $\det A\ne 0$. Let $A_i$ be $A$ with column $i$ replaced by $B$. Then

$$x_i = \frac{\det A_i}{\det A}.$$

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.
– William Blake (The Marriage of Heaven and Hell)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 47 | 18 | 76 | 29 | 93 | 85 | 34 | 61 | 52 |
| 86 | 11 | 57 | 28 | 70 | 39 | 94 | 45 | 02 | 63 |
| 95 | 80 | 22 | 67 | 38 | 71 | 49 | 56 | 13 | 04 |
| 59 | 96 | 81 | 33 | 07 | 48 | 72 | 60 | 24 | 15 |
| 73 | 69 | 90 | 82 | 44 | 17 | 58 | 01 | 35 | 26 |
| 68 | 74 | 09 | 91 | 83 | 55 | 27 | 12 | 46 | 30 |
| 37 | 08 | 75 | 19 | 92 | 84 | 66 | 23 | 50 | 41 |
| 14 | 25 | 36 | 40 | 51 | 62 | 03 | 77 | 88 | 99 |
| 21 | 32 | 43 | 54 | 65 | 06 | 10 | 89 | 97 | 78 |
| 42 | 53 | 64 | 05 | 16 | 20 | 31 | 98 | 79 | 87 |

The Fibonacci number system: Every integer $n$ has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where $k_i \ge k_{i+1}+2$ for all $i$, $1\le i < m$ and $k_m \ge 2$.

## Fibonacci Numbers

$$1,1,2,3,5,8,13,21,34,55,89,\ldots$$

Definitions:

$$F_i = F_{i-1}+F_{i-2}, \quad F_0=F_1=1,$$
$$F_{-i}=(-1)^{i-1}F_i,$$
$$F_i = \frac{1}{\sqrt{5}}\left(\phi^i - \hat{\phi}^i\right),$$

Cassini's identity: for $i > 0$:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$F_{n+k} = F_k F_{n+1} + F_{k-1}F_n,$$
$$F_{2n} = F_n F_{n+1} + F_{n-1}F_n.$$

Calculation by matrices:

$$\begin{pmatrix}F_{n-2} & F_{n-1}\\ F_{n-1} & F_n\end{pmatrix} = \begin{pmatrix}0 & 1\\ 1 & 1\end{pmatrix}^n.$$