Team Reference Document
Team #define true false, TU München
NWERC 2014

Inhaltsverzeichnis

IO

C++ Input/Output/Limits

```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <limits>
#include <algorithm>
#include <math.h>
#include <cstdlib>

#include <queue>
#include <vector>
#include <set>
#include <map>
#include <unordered_map>
#include <unordered_set>

using namespace std;
const int iMAX = numeric_limits<int>::max();
const int iMIN = numeric_limits<int>::min();
typedef long long LL;

int main() {
    // massively improve cout and cin performance for large streams
    ios::sync_with_stdio(false);
    cin.tie(0);

    // 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;

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

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

Computations

Greatest Common Divisor

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

Binomial Coefficients

```
long binomial(long n, long k) {
    if (k > n - k) return binomial(n, n - k);
    long result = 1;
    if (k > n) return 0;
    for (long next = 1; next <= k; ++next) {
        long cancelled = gcd(result, next);
        result = (result / cancelled) * (n - next + 1);
        result /= next / cancelled;
    }
    return result;
}
```

Data Structures

Union Find

```
initialize(): for all x, boss[x] = x, rank[x] = 0.

union(x, y)
    a = find(x); b = find(y);
    if (rank(a) < rank(b)) boss[a] = b;
    if (rank(a) > rank(b)) boss[b] = a;
    if (rank(a) == rank(b)) {boss[b] = a; rank[a] += 1;}

find(x)
    if (boss[x] == x) return x;
    boss[x] = find(boss[x]); // path compression
    return boss[x];
```

Math-Stuff

Euclid-Stuff

```
// 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) {
    int tmp;
    while(b){a%=b; tmp=a; a=b; b=tmp;}
    return a;
}

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

// returns d = 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 solutions;
    int d = extended_euclid(a, n, x, y);
    if (!(b%d)) {
        x = mod (x*(b/d), n);
        for (int i = 0; i < d; i++)
            solutions.push_back(mod(x + i*(n/d), n));
    }
    return solutions;
}

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

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
```

```
67  // Return (z,M).  On failure , M = −1.
68  PII chinese_remainder_theorem(int x, int a, int y, int b) {
69    int s, t;
70    int d = extended_euclid(x, y, s, t);
71    if (a%d != b%d) return make_pair(0, −1);
72    return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
73  }
74
75  // Chinese remainder theorem: find z such that
76  // z % x[i] = a[i] for all i.  Note that the solution is
77  // unique modulo M = lcm_i (x[i]).  Return (z,M).  On
78  // failure , M = −1.  Note that we do not require the a[i]'s
79  // to be relatively prime.
80  PII chinese_remainder_theorem(const VI &x, const VI &a) {
81    PII ret = make_pair(a[0], x[0]);
82    for (int i = 1; i < x.size(); i++) {
83      ret = chinese_remainder_theorem(ret.second, ret.first, x[i], a[i]);
84      if (ret.second == −1) break;
85    }
86    return ret;
87  }
88
89  // computes x and y such that ax + by = c; on failure , x = y =−1
90  void linear_diophantine(int a, int b, int c, int &x, int &y) {
91    int d = gcd(a,b);
92    if (c%d) {
93      x = y = −1;
94    } else {
95      x = c/d * mod_inverse(a/d, b/d);
96      y = (c−a*x)/b;
97    }
98  }
99
100 int main() {
101
102   // expected: 2
103   cout << gcd(14, 30) << endl;
104
105   // expected: 2 −2 1
106   int x, y;
107   int d = extended_euclid(14, 30, x, y);
108   cout << d << " " << x << " " << y << endl;
109
110   // expected: 95 45
111   VI sols = modular_linear_equation_solver(14, 30, 100);
112   for (int i = 0; i < (int) sols.size(); i++) cout << sols[i] << " ";
113   cout << endl;
114
115   // expected: 8
116   cout << mod_inverse(8, 9) << endl;
117
118   // expected: 23 56
119   //           11 12
120   int xs[] = {3, 5, 7, 4, 6};
121   int as[] = {2, 3, 2, 3, 5};
```

```
122   PII ret = chinese_remainder_theorem(VI (xs, xs+3), VI(as, as+3));
123   cout << ret.first << " " << ret.second << endl;
124   ret = chinese_remainder_theorem (VI(xs+3, xs+5), VI(as+3, as+5));
125   cout << ret.first << " " << ret.second << endl;
126
127   // expected: 5 −15
128   linear_diophantine(7, 2, 5, x, y);
129   cout << x << " " << y << endl;
130 }
```

Gauss-Jordan

```
1   // Gauss−Jordan elimination with full pivoting.
2   //
3   // Uses:
4   //    (1) solving systems of linear equations (AX=B)
5   //    (2) inverting matrices (AX=I)
6   //    (3) computing determinants of square matrices
7   //
8   // Running time: O(n^3)
9   //
10  // INPUT:    a[][] = an nxn matrix
11  //           b[][] = an nxm matrix
12  //
13  // OUTPUT:    X     = an nxm matrix (stored in b[][])
14  //           A^{−1} = an nxn matrix (stored in a[][])
15  //            returns determinant of a[][]
16
17  #include <iostream>
18  #include <vector>
19  #include <cmath>
20
21  using namespace std;
22
23  const double EPS = 1e−10;
24
25  typedef vector<int> VI;
26  typedef double T;
27  typedef vector<T> VT;
28  typedef vector<VT> VVT;
29
30  T GaussJordan(VVT &a, VVT &b) {
31    const int n = a.size();
32    const int m = b[0].size();
33    VI irow(n), icol(n), ipiv(n);
34    T det = 1;
35
36    for (int i = 0; i < n; i++) {
37      int pj = −1, pk = −1;
38      for (int j = 0; j < n; j++) if (!ipiv[j])
39        for (int k = 0; k < n; k++) if (!ipiv[k])
40      if (pj == −1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
41      if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
42      ipiv[pk]++;
43      swap(a[pj], a[pk]);
```

```
44      swap(b[pj], b[pk]);
45      if (pj != pk) det *= -1;
46      irow[i] = pj;
47      icol[i] = pk;
48
49      T c = 1.0 / a[pk][pk];
50      det *= a[pk][pk];
51      a[pk][pk] = 1.0;
52      for (int p = 0; p < n; p++) a[pk][p] *= c;
53      for (int p = 0; p < m; p++) b[pk][p] *= c;
54      for (int p = 0; p < n; p++) if (p != pk) {
55        c = a[p][pk];
56        a[p][pk] = 0;
57        for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
58        for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
59      }
60    }
61
62    for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
63      for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
64    }
65
66    return det;
67 }
68
69 int main() {
70    const int n = 4;
71    const int m = 2;
72    double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
73    double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
74    VVT a(n), b(n);
75    for (int i = 0; i < n; i++) {
76      a[i] = VT(A[i], A[i] + n);
77      b[i] = VT(B[i], B[i] + m);
78    }
79
80    double det = GaussJordan(a, b);
81
82    // expected: 60
83    cout << "Determinant: " << det << endl;
84
85    // expected: -0.233333 0.166667 0.133333 0.0666667
86    //            0.166667 0.166667 0.333333 -0.333333
87    //            0.233333 0.833333 -0.133333 -0.0666667
88    //            0.05 -0.75 -0.1 0.2
89    cout << "Inverse: " << endl;
90    for (int i = 0; i < n; i++) {
91      for (int j = 0; j < n; j++)
92        cout << a[i][j] << ' ';
93      cout << endl;
94    }
95
96    // expected: 1.63333 1.3
97    //           -0.166667 0.5
98    //            2.36667 1.7
99    //           -1.85 -1.35
100   cout << "Solution: " << endl;
101   for (int i = 0; i < n; i++) {
102     for (int j = 0; j < m; j++)
103       cout << b[i][j] << ' ';
104     cout << endl;
105   }
106 }
```

Collected Binomials

```
1  //Berechnet alle Binomialkoeffizienten (n ueber k) mod m mit n<N
2  int binom[N][N];
3  void calcbinomials(int m) {
4    for(int n=0; n<N; n++) {
5      binom[n][0] = binom[n][n] = 1;
6      for(int k=1; k<n; k++)
7        binom[n][k] = (binom[n-1][k]+binom[n-1][k-1])%m;
8    }
9  }
10 //Berechnet einzelnen Binomialkoeffizienten in Restklasse O(log n)
11 void calcbinom(int n, int k, int m) {
12   return (fak[n] * inverse(fak[k], m) * inverse(fak[n-k], m))%m;
13 } //fak[n] = (n!)%m
14
15
16 //Berechnet fuer fixes n fuer alle k (n ueber k)  O(n)
17 void calcbinomrow(int n) {
18   binom[n][0] = 1;
19   for(int k=1; k<=n; k++) {
20     binom[n][k] = binom[n][k-1]*(n-k+1)/k; //*inv(k) % MOD
21   }
22 }
```

Shortest Paths

Floyd-Warshall

Floyd-Warshall kommt mit negativen Gewichten zurecht. All sources, all targets.

```
1  procedure FloydWarshallWithPathReconstruction ()
2      for k := 1 to n
3          for i := 1 to n
4              for j := 1 to n
5                  if (path[i][k] + path[k][j] < path[i][j]) {
6                      path[i][j] := path[i][k]+path[k][j];
7                      next[i][j] := next[i][k];
8                  }
9
10  function Path (i,j)
11     if path[i][j] equals infinity then
12         return "no path";
13     int intermediate := next[i][j];
14     if intermediate equals 'null' then
15         return " ";
16     else
```

```
17          return Path(i,intermediate)
18              + intermediate
19              + Path(intermediate,j);
```

Dijkstra/Java

```java
1   PriorityQueue<Item> q = new PriorityQueue<Item>();
2
3   Item[] index = new Item[n];
4   for(int i = 0; i < n; ++i) index[i] = new Item(-1, oo);
5
6   index[start] = new Item(-1, 0);
7   q.add(new Item(start, 0));
8
9   while(!q.isEmpty()) {
10      Item curr = q.poll();
11      if(curr.value > index[curr.node].value) continue;
12      /*if(curr.node == end) break; */
13      ArrayList<Item> edges = v.get(curr.node);
14      for(int i = 0; i < edges.size(); ++i) {
15          int nv = edges.get(i).value + curr.value;
16          int otherNode = edges.get(i).node;
17          Item oi = index[otherNode];
18          if(nv < oi.value) {
19              oi.value = nv;
20              oi.node = curr.node;
21              q.add(new Item(otherNode, nv));
22          }
23      }
24  }
25  return index;
```

```java
21  int phase = 0;
22  int nextPhaseStart = -1;
23  q.add(0);
24  boolean jackpot = false; // neg cycle
25  while(!q.isEmpty()) {
26      int i = q.poll();
27      inQueue[i] = false;
28      if(i == nextPhaseStart) {
29          phase++;
30          nextPhaseStart = -1;
31      }
32      if(phase == n-1) {
33          System.out.format("Case \#%d: Jackpot\n", numCase+1);
34          jackpot = true;
35          break;
36      }
37      Item it = index[i];
38      ArrayList<Item> e = v.get(i);
39      for(int x = 0; x < e.size(); ++x) {
40          Item edge = e.get(x);
41          double nv = edge.value + it.value;
42          Item other = index[edge.node];
43          if(nv < other.value) {
44              other.value = nv;
45              if(!inQueue[edge.node]) {
46                  q.add(edge.node);
47                  if(nextPhaseStart == -1) nextPhaseStart = edge.node;
48                  inQueue[edge.node] = true;
49              }
50          }
51      }
52  }
```

Bellman-Ford/Java

```java
1   static class Item {
2       public int node;
3       public double value;
4   }
5
6   ArrayList<ArrayList<Item>> v = new ArrayList<ArrayList<Item>>(n);
7   for(int i = 0; i < n; ++i) {
8       v.add(new ArrayList<Item>());
9   }
10  // Kanten einfuegen:
11  // v.get(a).add(new Item(b, c));
12  ArrayDeque<Integer> q = new ArrayDeque<Integer>();
13  Item[] index = new Item[n];
14  index[0] = new Item(-1, 0);
15  for(int i = 1; i < n; ++i) {
16      index[i] = new Item(-1, oo);
17  }
18
19  boolean[] inQueue = new boolean[n];
20  inQueue[0] = true;
```

Flow

MaxFlow Push-Relabel

```cpp
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]]++;
```

```cpp
    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;
  }
};
```

Matching

Max Bipartite Matching

```cpp
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;
```

```
16  }
17
18  int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
19    mr = VI(w.size(), -1);
20    mc = VI(w[0].size(), -1);
21
22    int ct = 0;
23    for (int i = 0; i < w.size(); i++) {
24      VI seen(w[0].size());
25      if (FindMatch(i, w, mr, mc, seen)) ct++;
26    }
27    return ct;
28  }
```

Graph Stuff

Strongly Connected Components

```
1   // Der Graph.
2   vector<int> g[20000];
3   // Anzahl der Knoten im Graphen.
4   int V;
5
6   // Interne Variablen fuer den Algorithmus
7   int d[20000], low[20000];
8   int t;
9   vector<int> stack;
10  bool instack[20000];
11
12  // Ergebnis-Struktur: enthaelt am Ende die starken
13  //Zusammenhangskomponenten (als Listen von Knotenindizes)
14  vector<vector<int> > sccs;
15
16  void VISIT(int v) {
17    d[v] = low[v] = ++t;
18    stack.push_back(v);
19    instack[v] = true;
20    for (vector<int>::iterator w = g[v].begin(); w != g[v].end(); ++w) {
21      if (! d[*w]) {
22        VISIT(*w);
23        low[v] = min(low[v], low[*w]);
24      } else if (instack[*w]) {
25        low[v] = min(low[v], low[*w]);
26      }
27    }
28
29    if (d[v] == low[v]) {
30      vector<int> scc;
31      while(1) {
32        int w = stack.back();
33        stack.pop_back();
34        instack[w] = false;
35        scc.push_back(w);
36        if (v == w)
37          break;
```

```
38      }
39      sccs.push_back(scc);
40    }
41  }
42
43  // Aufruf der VISIT Funktion:
44  memset(d, 0, sizeof(d));
45  memset(instack, 0, sizeof(instack));
46  t = 0;
47  for (int v = 0; v < V; v++)
48    if (! d[v])
49      VISIT(v);
```

Topological Sort

```
1   void dsf(int x) {
2     if(visited[x] {
3       if(!f[x]) circle = true;
4       return;
5     }
6     visited[x] = true;
7
8     for(Integer curr : list.get(x)) dsf(curr);
9
10    out[tt] = x;
11    tt++;
12    f[x] = true;
13  }
```

Strings

Suffix Array

```
1   struct SuffixArray {
2     const int L;
3     string s;
4     vector<vector<int> > P;
5     vector<pair<pair<int,int>,int> > M;
6
7     SuffixArray(const string &s) : L(s.length()), s(s), P(1, vector<int>(L, 0)), M(L) {
8       for (int i = 0; i < L; i++) P[0][i] = int(s[i]);
9       for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
10        P.push_back(vector<int>(L, 0));
11        for (int i = 0; i < L; i++)
12          M[i] = make_pair(make_pair(P[level-1][i],
13                  i + skip < L ? P[level-1][i + skip] : -1000),
14              i);
15        sort(M.begin(), M.end());
16        for (int i = 0; i < L; i++)
17          P[level][M[i].second] = (i > 0 && M[i].first == M[i-1].first) ?
18            P[level][M[i-1].second] : i;
19      }
20    }
21
22    vector<int> GetSuffixArray() { return P.back(); }
```

```
23
24    // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
25    int LongestCommonPrefix(int i, int j) {
26      int len = 0;
27      if (i == j) return L - i;
28      for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--) {
29        if (P[k][i] == P[k][j]) {
30          i += 1 << k;
31          j += 1 << k;
32          len += 1 << k;
33        }
34      }
35      return len;
36    }
37 };
38
39 int main() {
40
41    // bobocel is the 0'th suffix
42    //  obocel is the 5'th suffix
43    //   bocel is the 1'st suffix
44    //    ocel is the 6'th suffix
45    //     cel is the 2'nd suffix
46    //      el is the 3'rd suffix
47    //       l is the 4'th suffix
48    SuffixArray suffix("bobocel");
49    vector<int> v = suffix.GetSuffixArray();
50
51    // Expected output: 0 5 1 6 2 3 4
52    //                  2
53    for (int i = 0; i < v.size(); i++) cout << v[i] << " ";
54    cout << endl;
55    cout << suffix.LongestCommonPrefix(0, 2) << endl;
56 }
```

Knuth-Morris-Pratt Algorithm

```
1  /* Searches for the string w in the string s (of length k). Returns the 0-based
2  index of the first match (k if no match is found).
3  Algorithm runs in O(k) time. */
4
5  #include <iostream>
6  #include <string>
7  #include <vector>
8
9  using namespace std;
10
11 typedef vector<int> VI;
12
13 void buildTable(string& w, VI& t)
14 {
15    t = VI(w.length());
16    int i = 2, j = 0;
17    t[0] = -1; t[1] = 0;
18
19    while(i < w.length())
20    {
21      if(w[i-1] == w[j]) { t[i] = j+1; i++; j++; }
22      else if(j > 0) j = t[j];
23      else { t[i] = 0; i++; }
24    }
25 }
26
27 int KMP(string& s, string& w)
28 {
29    int m = 0, i = 0;
30    VI t;
31
32    buildTable(w, t);
33    while(m+i < s.length())
34    {
35      if(w[i] == s[m+i])
36      {
37        i++;
38        if(i == w.length()) return m;
39      }
40      else
41      {
42        m += i-t[i];
43        if(i > 0) i = t[i];
44      }
45    }
46    return s.length();
47 }
48
49 int main()
50 {
51    string a = (string) "The example above illustrates the general technique for assembling "+
52      "the table with a minimum of fuss. The principle is that of the overall search: "+
53      "most of the work was already done in getting to the current position, so very "+
54      "little needs to be done in leaving it. The only minor complication is that the "+
55      "logic which is correct late in the string erroneously gives non-proper "+
56      "substrings at the beginning. This necessitates some initialization code.";
57
58    string b = "table";
59
60    int p = KMP(a, b);
61    cout << p << ": " << a.substr(p, b.length()) << " " << b << endl;
62 }
```

Geometry

Geometry/C++

```
1  double INF = 1e100;
2  double EPS = 1e-12;
3
4  struct PT {
5    double x, y;
6    PT() {}
```

```cpp
  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) {
  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
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 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;
```

```cpp
}

// 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;
```

```cpp
      // 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;

      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;
}
```

Geometry/Java

```java
P cross(P o) {
    return new P(y*o.z-z*o.y, z*o.x-x*o.z, x*o.y-y*o.x);
}

P scalar(P o) {
    return new P(x*o.x, y * o.y, z * o.z);
}

P r90() {
    return new P(-y, x, z);
}

P parallel(P p) {
    return cross(zeroOne).cross(p);
}

Point2D getPoint() {
    return new Point2D.Double(x / z, y / z);
}

static double computePolygonArea(ArrayList<Point2D.Double> points) {
    Point2D.Double[] pts = points.toArray(new Point2D.Double[points.size()]);
    double area = 0;
    for (int i = 0; i < pts.length; i++) {
        int j = (i+1) % pts.length;
        area += pts[i].x * pts[j].y - pts[j].x * pts[i].y;
    }
    return Math.abs(area)/2;
}
```

Graham Scan – Konvexe Huelle

1. Finde p_0 mit min y, Unentschieden: betrachte x
2. Sortiere $p_{1...n}$. $p_i < p_j = ccw(p_0, p_i, p_j)$
   (colinear → naechster zuerst)
3. Setze $p_{n+1} = p_0$
4. Push(p_0); Push(p_1); Push(p_2);
5. for $i = 3$ to $n + 1$

   (a) Solange Winkel der letzten zwei des Stacks und p_i rechtskurve: Pop()

   (b) Push(p_i)

```java
int minPoint = 0;
for(int i = 1; i < n; ++i) {
    if(points[i].y < points[minPoint].y ||
        (points[i].y == points[minPoint].y &&
            points[i].x < points[minPoint].x)) {
    minPoint = i;
    }
}
final int mx = points[minPoint].x;
final int my = points[minPoint].y;
Arrays.sort(points, new Comparator<Point>() {
    @Override
    public int compare(Point a, Point b) {
        int ccw = Line2D.relativeCCW(mx, my, a.x, a.y, b.x, b.y);
        if(ccw == 0 || Line2D.relativeCCW(mx, my, b.x, b.y, a.x, a.y) == 0) {
            // gleich...
            double d1 = a.distance(mx, my);
            double d2 = b.distance(mx, my);
            if((d2 < d1 && d2 != 0) || d1 == 0) {
                return 1;
            } else {
                return -1;
            }
        } else if(ccw == 1) {
            // clockwise... -> zuerst b -> a > b
            return 1;
        } else if(ccw == -1) {
            return -1;
        } else {
            System.out.println("shouldnt happen");
            System.exit(1);
        }
        return 0;
    }
});

ArrayList<Integer> stack = new ArrayList<Integer>();
stack.add(n-1);
for(int i = 0; i < n; ++i) {
    if(stack.size() < 2) {
        stack.add(i);
        continue;
    }
    int last = stack.get(stack.size() - 1);
    int l2 = stack.get(stack.size() - 2);
    int ccw = Line2D.relativeCCW(points[l2].x, points[l2].y,
     points[last].x, points[last].y, points[i].x, points[i].y);
    if(ccw != -1) {
        // clockwise oder gleiche Linie
        stack.remove(stack.size() - 1);
        i--;
    } else {
        stack.add(i);
    }
}
```

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

typedef double T;

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

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]);
```

```
52    vector<triple> tri = delaunayTriangulation(x, y);
53
54    //expected: 0 1 3
55    //          0 3 2
56
57    int i;
58    for(i = 0; i < tri.size(); i++)
59        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
60    return 0;
61 }
```

Trees

Binary Indexed Tree

```
1
2  //binary indexed tree
3  //verwaltet kumultative Summen in log(n)
4
5  int tree[1<<N];
6  int MaxVal = (1<<N)-1;
7
8  int readsum(int idx){//sum_{i in [1;idx]} f[i]
9      int sum = 0;
10     while (idx > 0){
11         sum += tree[idx];
12         idx -= (idx & -idx);
13     }
14     return sum;
15 }
16
17 int suminrange(int a, int b) { //sum_{i in [a;b[} f[i]
18     return readsum(b-1)-readsum(a-1);
19 }
20
21 void update(int idx ,int val){ //updates f[idx]->val
22     while (idx <= MaxVal){
23         tree[idx] += val;
24         idx += (idx & -idx);
25     }
26 }
```

Segment Tree- TODO

```
1  TODO
```

KD-tree

```
1  // ------------------------------------------------------------------
2  // A straightforward, but probably sub-optimal KD-tree implmentation that's
3  // probably good enough for most things (current it's a 2D-tree)
4  //
5  //  - constructs from n points in O(n lg^2 n) time
6  //  - handles nearest-neighbor query in O(lg n) if points are well distributed
7  //  - worst case for nearest-neighbor may be linear in pathological case
8  //
9  // Sonny Chan, Stanford University, April 2009
10 // ------------------------------------------------------------------
11
12 // number type for coordinates, and its maximum value
13 typedef long long ntype;
14 const ntype sentry = numeric_limits<ntype>::max();
15
16 // point structure for 2D-tree, can be extended to 3D
17 struct point {
18     ntype x, y;
19     point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
20 };
21
22 bool operator==(const point &a, const point &b)
23 {
24     return a.x == b.x && a.y == b.y;
25 }
26
27 // sorts points on x-coordinate
28 bool on_x(const point &a, const point &b)
29 {
30     return a.x < b.x;
31 }
32
33 // sorts points on y-coordinate
34 bool on_y(const point &a, const point &b)
35 {
36     return a.y < b.y;
37 }
38
39 // squared distance between points
40 ntype pdist2(const point &a, const point &b)
41 {
42     ntype dx = a.x-b.x, dy = a.y-b.y;
43     return dx*dx + dy*dy;
44 }
45
46 // bounding box for a set of points
47 struct bbox
48 {
49     ntype x0, x1, y0, y1;
50
51     bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}
52
53     // computes bounding box from a bunch of points
54     void compute(const vector<point> &v) {
55         for (int i = 0; i < v.size(); ++i) {
56             x0 = min(x0, v[i].x);   x1 = max(x1, v[i].x);
57             y0 = min(y0, v[i].y);   y1 = max(y1, v[i].y);
58         }
59     }
60
61     // squared distance between a point and this bbox, 0 if inside
```

```cpp
    ntype distance(const point &p) {                              // divide by taking half the array for each child
        if (p.x < x0) {                                           // (not best performance if many duplicates in the middle)
            if (p.y < y0)        return pdist2(point(x0, y0), p);  int half = vp.size()/2;
            else if (p.y > y1)   return pdist2(point(x0, y1), p);  vector<point> vl(vp.begin(), vp.begin()+half);
            else                 return pdist2(point(x0, p.y), p); vector<point> vr(vp.begin()+half, vp.end());
        }                                                          first = new kdnode();    first->construct(vl);
        else if (p.x > x1) {                                       second = new kdnode();  second->construct(vr);
            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 {                                   // simple kd-tree class to hold the tree and handle queries
            if (p.y < y0)        return pdist2(point(p.x, y0), p);struct kdtree
            else if (p.y > y1)   return pdist2(point(p.x, y1), p);{
            else                 return 0;           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());
// stores a single node of the kd-tree, either internal or leaf
                                                              root = new kdnode();
struct kdnode                                                 root->construct(v);
{                                                         }
    bool leaf;        // true if this is a leaf node (has one point)~kdtree() { delete root; }
    point pt;         // the single point of this is a leaf
    bbox bound;       // bounding box for set of points in children// recursive search method returns squared distance to nearest point
                                                          ntype search(kdnode *node, const point &p)
    kdnode *first, *second; // two children of this kd-node{
                                                              if (node->leaf) {
    kdnode() : leaf(false), first(0), second(0) {}            // commented special case tells a point not to find itself
    ~kdnode() { if (first) delete first; if (second) delete second; }//  if (p == node->pt) return sentry;
                                                          //     else
    // intersect a point with this node (returns squared distance)//         return pdist2(p, node->pt);
    ntype intersect(const point &p) {                         }
        return bound.distance(p);
    }                                                         ntype bfirst = node->first->intersect(p);
                                                              ntype bsecond = node->second->intersect(p);
    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp)                         // choose the side with the closest bounding box to search first
    {                                                         // (note that the other side is also searched if needed)
        // compute bounding box for points at this node      if (bfirst < bsecond) {
        bound.compute(vp);                                        ntype best = search(node->first, p);
                                                                  if (bsecond < best)
        // if we're down to one point, then we're a leaf node         best = min(best, search(node->second, p));
        if (vp.size() == 1) {                                     return best;
            leaf = true;                                      }
            pt = vp[0];                                       else {
        }                                                         ntype best = search(node->second, p);
        else {                                                    if (bfirst < best)
            // split on x if the bbox is wider than high (not best heuristic...)   best = min(best, search(node->first, p));
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)           return best;
                sort(vp.begin(), vp.end(), on_x);             }
            // otherwise split on y-coordinate               }
            else
                sort(vp.begin(), vp.end(), on_y);        // squared distance to the nearest
                                                         ntype nearest(const point &p) {
```

```
172          return search(root, p);
173      }
174 };
175
176 // ------------------------------------------------------------
177 // some basic test code here
178
179 int main()
180 {
181     // generate some random points for a kd-tree
182     vector<point> vp;
183     for (int i = 0; i < 100000; ++i) {
184         vp.push_back(point(rand()%100000, rand()%100000));
185     }
186     kdtree tree(vp);
187
188     // query some points
189     for (int i = 0; i < 10; ++i) {
190         point q(rand()%100000, rand()%100000);
191         cout << "Closest squared distance to (" << q.x << ", " << q.y << ")"
192              << " is " << tree.nearest(q) << endl;
193     }
194
195     return 0;
196 }
197
198 // ------------------------------------------------------------
```

Misc

Longest Increasing Subsequence

```
1  // Given a list of numbers of length n, this routine extracts a
2  // longest increasing subsequence.
3  //
4  // Running time: O(n log n)
5  //
6  //   INPUT: a vector of integers
7  //   OUTPUT: a vector containing the longest increasing subsequence
8  typedef vector<int> VI;
9  typedef pair<int,int> PII;
10 typedef vector<PII> VPII;
11
12 #define STRICTLY_INCREASNG
13
14 VI LongestIncreasingSubsequence(VI v) {
15     VPII best;
16     VI dad(v.size(), -1);
17
18     for (int i = 0; i < v.size(); i++) {
19 #ifdef STRICTLY_INCREASNG
20         PII item = make_pair(v[i], 0);
21         VPII::iterator it = lower_bound(best.begin(), best.end(), item);
22         item.second = i;
23 #else
```

```
24         PII item = make_pair(v[i], i);
25         VPII::iterator it = upper_bound(best.begin(), best.end(), item);
26 #endif
27         if (it == best.end()) {
28             dad[i] = (best.size() == 0 ? -1 : best.back().second);
29             best.push_back(item);
30         } else {
31             dad[i] = dad[it->second];
32             *it = item;
33         }
34     }
35
36     VI ret;
37     for (int i = best.back().second; i >= 0; i = dad[i])
38         ret.push_back(v[i]);
39     reverse(ret.begin(), ret.end());
40     return ret;
41 }
```

Simulated Annealing

```
1  Random r = new Random();
2  int numChanges = 0;
3  double T = 10000;
4  double alpha = 0.99;
5  int decreaseAfter = 20;
6  int nChanges = 0;
7  for(int i = 0; i < 1000000; ++i) {
8      // calculate newCost (apply 2-opt-step) (swap two things)
9      double delta = newCost - cost;
10     boolean accept = newCost <= cost;
11     if(!accept) {
12         double R = r.nextDouble();
13         double calc = Math.exp(-delta / T);
14         double maxDiff = Math.exp(-10000/T);
15         if(calc < maxDiff && i < 1000000/2) {
16             calc = maxDiff;
17         }
18         //System.out.println(calc);
19         if(calc > R) {
20             accept = true;
21         }
22     }
23     // if(i % 10000 == 0) {
24     //   System.out.println("after " + i + ": " + T);
25     // }
26
27     if(nChanges >= decreaseAfter) {
28         nChanges = 0;
29         T = alpha * T;
30     }
31     if(accept) {
32         cost = newCost;
33         numChanges++;
34         nChanges++;
```

```
35        } else {
36            // swap back
37            swap(trip, a, b);
38        }
39 }
```

Simplex Algorithm

```
1  // Two-phase simplex algorithm for solving linear programs of the form
2  //
3  //       maximize      c^T x
4  //       subject to    Ax <= b
5  //                     x >= 0
6  //
7  // INPUT: A -- an m x n matrix
8  //        b -- an m-dimensional vector
9  //        c -- an n-dimensional vector
10 //        x -- a vector where the optimal solution will be stored
11 //
12 // OUTPUT: value of the optimal solution (infinity if unbounded
13 //         above, nan if infeasible)
14 //
15 // To use this code, create an LPSolver object with A, b, and c as
16 // arguments.  Then, call Solve(x).
17 typedef long double DOUBLE;
18 typedef vector<DOUBLE> VD;
19 typedef vector<VD> VVD;
20 typedef vector<int> VI;
21
22 const DOUBLE EPS = 1e-9;
23
24 struct LPSolver {
25     int m, n;
26     VI B, N;
27     VVD D;
28
29     LPSolver(const VVD &A, const VD &b, const VD &c) :
30         m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, VD(n+2)) {
31         for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
32         for (int i = 0; i < m; i++) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
33         for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
34         N[n] = -1; D[m+1][n] = 1;
35     }
36
37     void Pivot(int r, int s) {
38         for (int i = 0; i < m+2; i++) if (i != r)
39             for (int j = 0; j < n+2; j++) if (j != s)
40                 D[i][j] -= D[r][j] * D[i][s] / D[r][s];
41         for (int j = 0; j < n+2; j++) if (j != s) D[r][j] /= D[r][s];
42         for (int i = 0; i < m+2; i++) if (i != r) D[i][s] /= -D[r][s];
43         D[r][s] = 1.0 / D[r][s];
44         swap(B[r], N[s]);
45     }
46
47     bool Simplex(int phase) {
48         int x = phase == 1 ? m+1 : m;
49         while (true) {
50             int s = -1;
51             for (int j = 0; j <= n; j++) {
52                 if (phase == 2 && N[j] == -1) continue;
53                 if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s]) s = j;
54             }
55             if (D[x][s] >= -EPS) return true;
56             int r = -1;
57             for (int i = 0; i < m; i++) {
58                 if (D[i][s] <= 0) continue;
59                 if (r == -1 || D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s] ||
60                     D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s] && B[i] < B[r]) r = i;
61             }
62             if (r == -1) return false;
63             Pivot(r, s);
64         }
65     }
66
67     DOUBLE Solve(VD &x) {
68         int r = 0;
69         for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r = i;
70         if (D[r][n+1] <= -EPS) {
71             Pivot(r, n);
72             if (!Simplex(1) || D[m+1][n+1] < -EPS) return -numeric_limits<DOUBLE>::infinity();
73             for (int i = 0; i < m; i++) if (B[i] == -1) {
74                 int s = -1;
75                 for (int j = 0; j <= n; j++)
76                     if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] < N[s]) s = j;
77                 Pivot(i, s);
78             }
79         }
80         if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
81         x = VD(n);
82         for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n+1];
83         return D[m][n+1];
84     }
85 };
86
87 int main() {
88     const int m = 4;
89     const int n = 3;
90     DOUBLE _A[m][n] = {
91         { 6, -1, 0 },
92         { -1, -5, 0 },
93         { 1, 5, 1 },
94         { -1, -5, -1 }
95     };
96     DOUBLE _b[m] = { 10, -4, 5, -5 };
97     DOUBLE _c[n] = { 1, -1, 0 };
98
99     VVD A(m);
100    VD b(_b, _b + m);
101    VD c(_c, _c + n);
102    for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);
```

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

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

Dates

```cpp
// 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.

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);
```

```cpp
  string day = intToDay (jd);

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

Primes

```cpp
// 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.
```

LatLon

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

```
20    return Q;
21  }
22
23  rect convert(ll& Q)
24  {
25    rect P;
26    P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
27    P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
28    P.z = Q.r*sin(Q.lat*M_PI/180);
29
30    return P;
31  }
32
33  int main()
```

```
34  {
35    rect A;
36    ll B;
37
38    A.x = -1.0; A.y = 2.0; A.z = -3.0;
39
40    B = convert(A);
41    cout << B.r << " " << B.lat << " " << B.lon << endl;
42
43    A = convert(B);
44    cout << A.x << " " << A.y << " " << A.z << endl;
45  }
```

<div align="center">

Theoretical Computer Science Cheat Sheet

</div>

Definitions		Series

$f(n) = O(g(n))$	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.** $\displaystyle\sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n},$

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.** $\left\{{n+1 \atop m+1}\right\} = \displaystyle\sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \displaystyle\sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k},$

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. $\left[{n+1 \atop m+1}\right] = \sum_k \left[{n \atop k}\right]\binom{k}{m} = \sum_{k=0}^{n}\left[{k \atop m}\right]n^{n-k} = n!\sum_{k=0}^{n}\frac{1}{k!}\left[{k \atop m}\right]$,   **39.** $\left[{x \atop x-n}\right] = \sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle\binom{x+k}{2n}$,

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

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

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

46. $\left\{{n \atop n-m}\right\} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[{m+k \atop k}\right]$,   **47.** $\left[{n \atop n-m}\right] = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{{m+k \atop k}\right\}$,

48. $\left\{{n \atop \ell+m}\right\}\binom{\ell+m}{\ell} = \sum_k \left\{{k \atop \ell}\right\}\left\{{n-k \atop m}\right\}\binom{n}{k}$,   **49.** $\left[{n \atop \ell+m}\right]\binom{\ell+m}{\ell} = \sum_k \left[{k \atop \ell}\right]\left[{n-k \atop m}\right]\binom{n}{k}$.

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
$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

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 $c = \tfrac{3}{2}$. 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 $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet

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

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

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

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	Matrices	More Trig.

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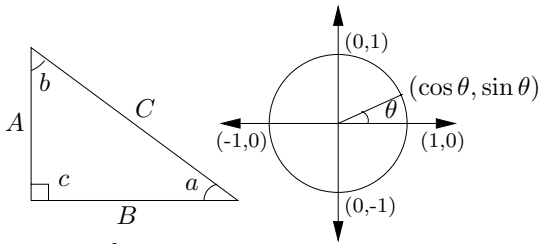
Pythagorean theorem:
$$C^2 = A^2 + B^2.$$

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.
$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \mathrm{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 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:
$$\mathrm{perm}\, A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

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

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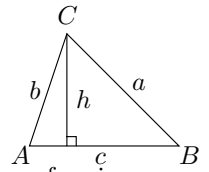$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
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	∞

... 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 $k \cdot c(G - S) \leq |S|$.
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 embedded 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 ≤ 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_∞ metric:
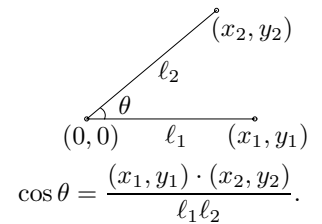
$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$
$$\left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p},$$
$$\lim_{p \to \infty} \left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p}.$$

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

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

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:

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

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