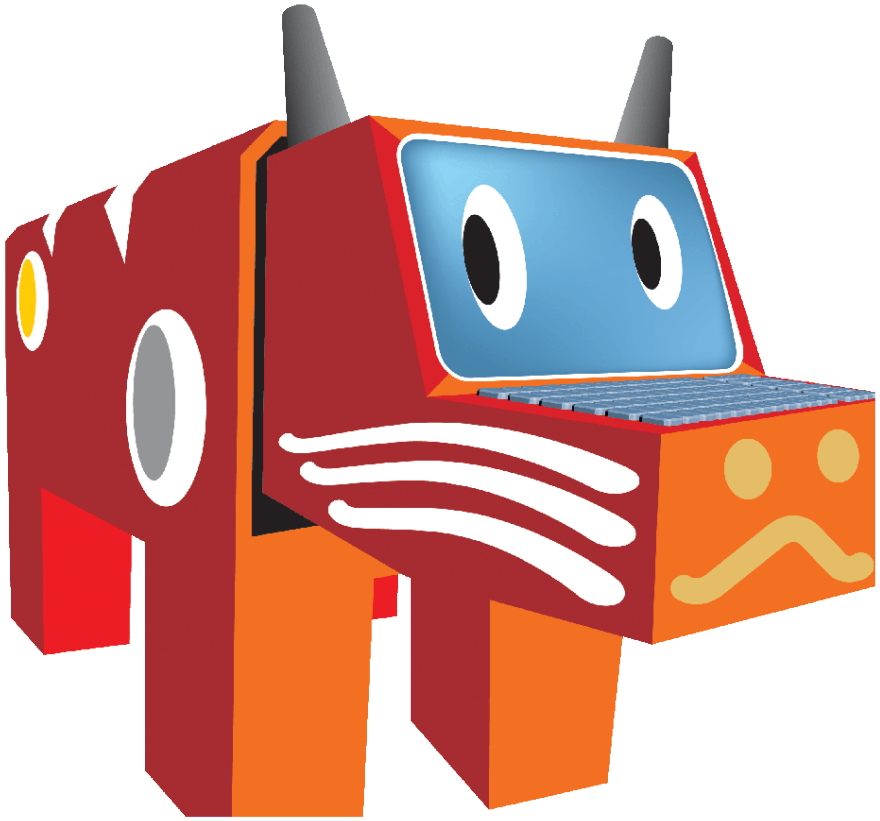
AIZU
THE UNIVERSITY OF AIZU

パソコン甲子園２０１５本選
プログラミング部門　解説



会津大学

概要

AIZU
THE UNIVERSITY OF AIZU

#	タイトル	分野	実装	思考	得点	色
1	直方体	整列・探索・判定	★☆	★☆	6	●
2	関連商品	データ構造・整列	★★	☆	6	○
3	虫食い算	探索	★★	★	6	●
4	貴金属リサイクル	計算	☆	★★	6	●
5	完全二国間貿易	データ構造	★	★★☆	8	●
6	プログラム停止判定	シミュレーション	★★★	★☆	10	●
7	スケジューラ	グラフ・データ構造	★★★☆	★★★	12	●
8	消える数列、消えない数列	動的計画法・構文解析	★☆	★★★☆	14	●
9	線分配置	データ構造・計算幾何	★★★☆	★★★★	16	●
10	あみだくじ	グラフ	★★★☆	★★★★☆	16	●

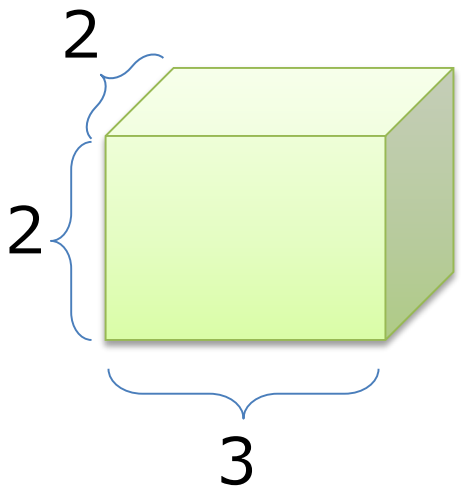
問１　直方体

AIZU
THE UNIVERSITY OF AIZU

問題概要

- 与えられた6枚の長方形で直方体が作れるか判定せよ.
- 各長方形の縦と横の長さが与えられる.

入力例１	出力例１
2　2 2　3 2　3 2　3 2　2 3　2	yes

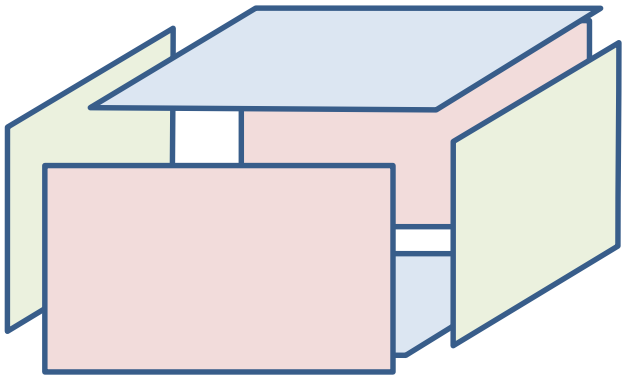


問１　直方体

AIZU
THE UNIVERSITY OF AIZU

出題のねらい

- 問われていること自体はシンプル.
- 様々なアルゴリズムが考えられる問題.
- どのような条件を満たせば、直方体を作れるのか考える.

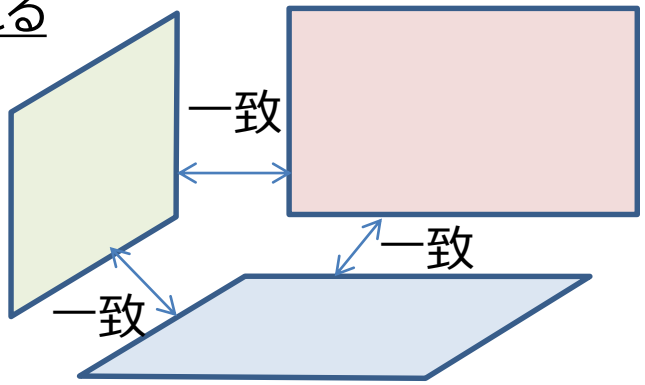


問１　直方体

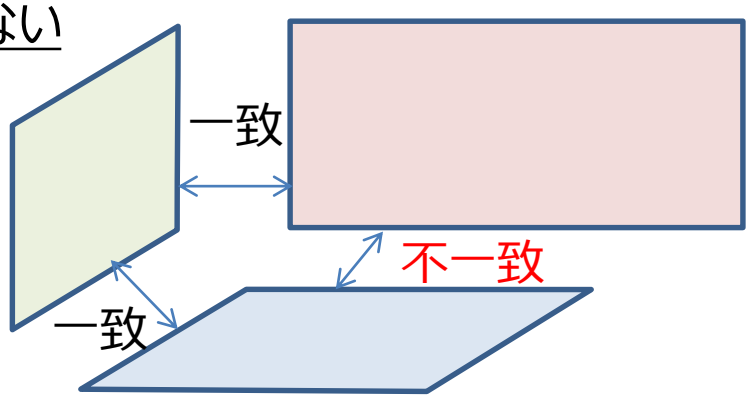
解法１

- 全ての長方形を使って、向かい合わせのペアが３つ作れるか判定（ペア１、ペア２、ペア３）.
  - 作れない場合は no　（作れる場合yesとは断定できない）.
- 各ペアの2通りの向き、3つのペアの位置関係を全て考慮して、ペア1、ペア2、ペア3の縦横の長さで、循環が作れるか判定する.

作れる



作れない

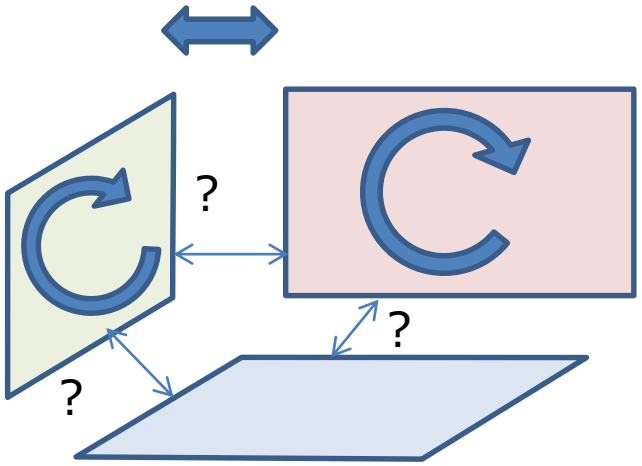


問１　直方体

AIZU
THE UNIVERSITY OF AIZU

解法１

- １つのペアを固定すると実装が楽になる.
- 2×２×２＝８通り.



問１　直方体

AIZU
THE UNIVERSITY OF AIZU

C++による解答例１

```
class Rect {
public:
  int m_w, m_h;
  Rect() {}
  bool operator == ( const Rect& r ) const {
    return r.m_w == m_w && r.m_h == m_h;
  }
  bool operator < ( const Rect& r ) const {//sort用
    if ( m_w < r.m_w ) return true;
    if ( m_w == r.m_w ) return m_h < r.m_h;
    return false;
  }
};

bool IsCycle( int a1, int a2, int a3, int a4, int a5, int a6 ) {
  return a1 == a6 && a2 == a3 && a4 == a5;
}
```

問１　直方体

AIZU
THE UNIVERSITY OF AIZU

C++による解答例1（つづき）

```
bool IsPossible( const Rect* aRect ) {
  //ペアが作れるか？
  if ( aRect[0] == aRect[1] && aRect[2] == aRect[3] && aRect[4] == aRect[5] ) {

    int c[6] = { aRect[0].m_w, aRect[0].m_h, aRect[2].m_w, aRect[2].m_h,
                 aRect[4].m_w, aRect[4].m_h };

    //サイクルになる組み合わせ方があるか？（３！）
    if ( IsCycle(c[0], c[1], c[2], c[3], c[4], c[5]) ||
         IsCycle(c[0], c[1], c[2], c[3], c[5], c[4]) ||
         IsCycle(c[0], c[1], c[3], c[2], c[4], c[5]) ||
         IsCycle(c[0], c[1], c[3], c[2], c[5], c[4]) ||
         IsCycle(c[1], c[0], c[2], c[3], c[4], c[5]) ||
         IsCycle(c[1], c[0], c[2], c[3], c[5], c[4]) ||
         IsCycle(c[1], c[0], c[3], c[2], c[4], c[5]) ||
         IsCycle(c[1], c[0], c[3], c[2], c[5], c[4]) ) {
      return true;
    }

  }
  return false;
}
```

問１　直方体

AIZU
THE UNIVERSITY OF AIZU

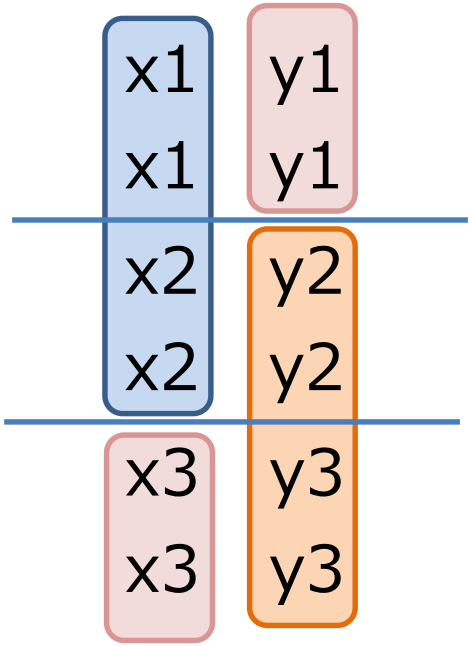
C++による解答例1（つづき）

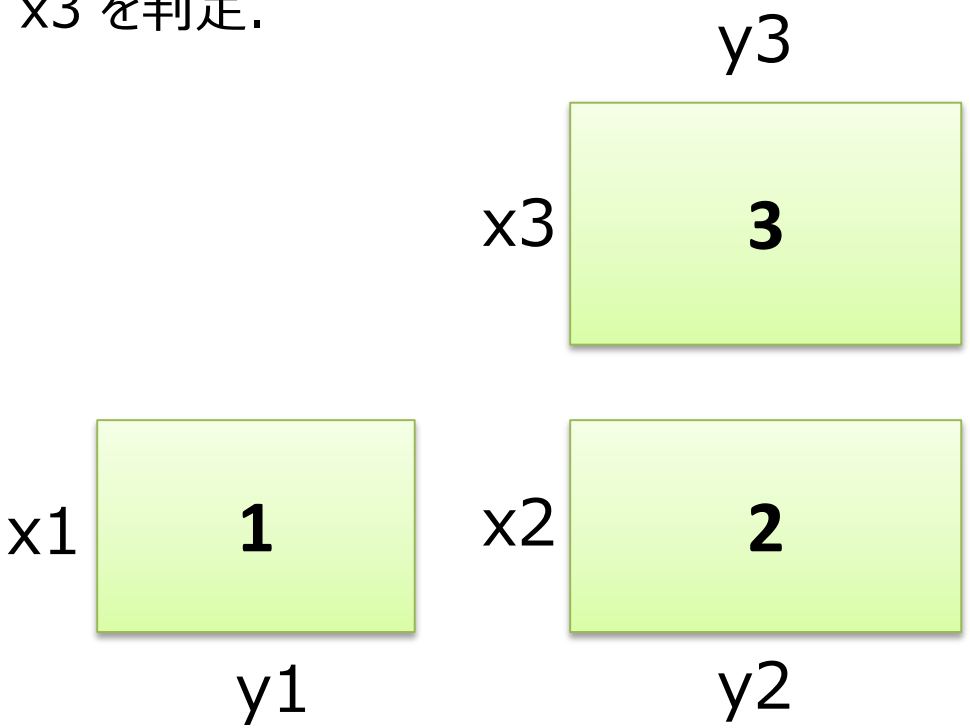
```
main() {

  Rect aRect[6];
  for ( int i=0; i<6; ++i )
    if ( scanf("%d %d", &aRect[i].m_h, &aRect[i].m_w) != 2 ) return 0;

  Rect aRectTmp[6];
  for ( int i=0; i<=(1<<6); ++i ) {//6ビットで縦横入れ替えるかどうかの全パターンを作る

    for ( int j=0; j<6; ++j ) {

      aRectTmp[j] = aRect[j];

      //iのjビット目が1なら j番目の長方形の縦と横を入れ替える
      if ( i&(1<<j) ) std::swap( aRectTmp[j].m_h, aRectTmp[j].m_w );
    }

    std::sort( aRectTmp, aRectTmp+6 );
    if ( IsPossible( aRectTmp ) ) {
      printf("yes¥n");
      return 0;
    }
  }
  printf("no¥n");}
```

問１　直方体

解法2

- ３つのペアを作る.
  - 全て横長にして、小さい順に整列しておく（高さ→幅の順）.
  - これを(x1, y1), (x2, y2), (x3, y3)とする.
- x1 = x2, y2 = y3, y1 = x3 を判定.





問１　直方体

AIZU
THE UNIVERSITY OF AIZU

C++による解答例3

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

bool solve(){
  pair<int, int> D[6];
  for ( int i = 0; i < 6; i++ ) {
    cin >> D[i].first >> D[i].second;
    if ( D[i].first > D[i].second) swap(D[i].first, D[i].second);
  }
  sort(D, D+6);
  pair<int, int> F[3];
  for ( int i = 0; i < 6; i+=2){
    if ( D[i] == D[i+1] ) D[i/2] = D[i];
    else return false;
  }

  return D[0].first == D[1].first && D[0].second == D[2].first && D[1].second == D[2].second;
}

main(){
  if ( solve() ) cout << "yes" << endl;
  else cout << "no" << endl;
}
```

問２　関連商品

AIZU
THE UNIVERSITY OF AIZU

問題概要

- 一緒に買われた商品のリストがいくつか与えられる.
  - 10個以内、100セットまで.
- 指定された回数 F 回以上、同時に買われた商品のペアを全て出力する.
- 出力の仕方は、商品ペア内で辞書順、さらに、商品ペア同士でも辞書順.

出題のねらい

- ある商品のペアについて、一緒に買われたかどうか判断する必要がある(多数ある、文字列ペア同士の比較）.
- 文字列のペアを管理できるかが問われている（実装力）.

問２　関連商品

AIZU
THE UNIVERSITY OF AIZU

解法１

- 買われた商品セットごとに、全ての商品のペアを作る.
  milk bread apple potato　なら
  ⇨ (apple bread), (apple milk), (apple potato),
  (bread milk), (bread potato), (milk potato)
  - このとき、単語ペアを辞書順にしておく.
- 何らかのデータ構造に文字列のペアを入れていく.
- すでにそのペアがあれば、カウント＋１．無ければデータ構造内に新しく追加する.
  - データ構造：　配列(C)、map(C++)、TreeMap(Java)など
- すでにペアがあるかどうかは、線形探索でも十分間に合う.

問２　関連商品

AIZU
THE UNIVERSITY OF AIZU

解答例1（C++）

```
int main() {
  for (int N, F; cin >> N >> F;) {
    map<pair<string, string>, int> m;
    for (int i = 0; i < N; ++i) {
      int M; cin >> M;
      vector<string> item(M);
      for (int j = 0; j < M; ++j) cin >> item[j];
      sort(item.begin(), item.end());
      for (int j = 0; j < M; ++j) {
        for (int k = j+1; k < M; ++k) {
          m[make_pair(item[j], item[k])] += 1;
        }
      }
    }
    int res = 0;
    for (map<pair<string, string>, int>::iterator it = m.begin(); it != m.end(); ++it) {
      if (it->second >= F) ++res;
    }
    cout << res << endl;
    for (map<pair<string, string>, int>::iterator it = m.begin(); it != m.end(); ++it) {
      if (it->second >= F) cout << it->first.first << " " << it->first.second << endl;
    }
  }
  return 0;
}
```

問２　関連商品

AIZU
THE UNIVERSITY OF AIZU

解答例2（Ｊａｖａ）

商品ペア型定義

```
class StringPair implements Comparable< StringPair > {

    public String first;
    public String second;

    StringPair( String first, String second ) {
        this.first = first;
        this.second = second;
    }

    public int compareTo( StringPair other ) {
        int cmp = first.compareTo(other.first);
        return cmp == 0 ? second.compareTo(other.second) : cmp;
    }

}
```

問２　関連商品

AIZU
THE UNIVERSITY OF AIZU

解答例2（Ｊａｖａ）

商品ペアの入れ物とインデクスの入れ物作成、商品ペア出現カウント初期化

```
class F02 {

    static final int MAX_N = 100;
    static final int MAX_M = 100;

    void solve(){

        TreeMap< StringPair, Integer > mapSSN =
                                new TreeMap< StringPair, Integer >();

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        int f = sc.nextInt();
        int count[] = new int[MAX_M*MAX_M*MAX_N];
        int g = 0;
        for ( int i=0; i<MAX_M*MAX_M*MAX_N; ++i ) count[i] = 0;
```

問２　関連商品

AIZU
THE UNIVERSITY OF AIZU

解答例2（Ｊａｖａ）

商品ペアの組み合わせを全てつくってカウント

```
for ( int i=0; i<n; ++i ) {
    int m = sc.nextInt();
    String goods[] = new String[m];
    for ( int j=0; j<m; ++j ) goods[j] = sc.next();

    for ( int j=0; j<m; ++j ) {
        for ( int k=j+1; k<m; ++k ) {
            StringPair pair = null;
            if ( goods[j].compareTo( goods[k] ) < 0 )
                pair = new StringPair( goods[j], goods[k] );
            else
                pair = new StringPair( goods[k], goods[j] );
            if ( mapSSN.containsKey( pair ) ) {
                ++count[mapSSN.get(pair)];
                if ( count[mapSSN.get(pair)] == f ) ++g;
            }
            else  {
                count[mapSSN.size()] = 1;
                mapSSN.put(pair, mapSSN.size());
                if ( f == 1 ) ++g;
            }
        }
    }
}
```

問２　関連商品

AIZU
THE UNIVERSITY OF AIZU

解答例2（Ｊａｖａ）

出現回数が指定回数以上なら出力

```
        System.out.println( g );
        Iterator<StringPair> itPair = mapSSN.keySet().iterator();
        while ( itPair.hasNext() ) {
            StringPair key = itPair.next();
            if ( count[mapSSN.get(key)] >= f ) {
                System.out.println(key.first + " " + key.second);
            }
        }

    }

    public static void main(String[] a) {new F02().solve(); }
}
```

問２　関連商品

AIZU
THE UNIVERSITY OF AIZU

解法２

- 商品 i と商品 j が一緒に買われた回数をテーブル T[i][j] に記録していく.
- 入力文字列 name を整数 id に変換できれば、高度なデータ構造がなくても実装が可能.
  - 問題のサイズが小さいので線形探索も使える.

問２　関連商品

解法２

```
int getID(string s){
  // 何らかの方法で文字列sから
  // 対応する番号idを返す
  // 例えば、
  //      配列と線形探索



}


string getName(int id){
  // 何らかの方法で番号idから
  // 対応する文字列sを返す



}
```

```
for ( int i = 0; i < N; i++ ){
   cin >> M;

   // 配列 v に M 個の商品を入力

   for ( int a = 0; a < M-1; a++ ){
     for ( int b = a+1; b < M; b++ ){
       int x = getID(v[a]);
       int y = getID(v[b]);
       T[x][y]++;
       T[y][x] = T[x][y];
     }
   }
 }

 for ( int i = 1; i <= MAX-1; i++ ){
   for ( int j = i+1; j <= MAX; j++ ){
     if ( T[i][j] >= F ) {
       // getName(i)とgetName(j)の組をリストに追加
     }
   }
 }

 // リストをソートして出力
```

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

問題概要

- 下のような形の筆算を満たすような、数字の埋め方が何通りあるかを求める.

```
      A
    B C
+ D E F
-------
  G H I
```

- このとき、ＡからＩには、１から９の数字が一度だけしか使えない.
- さらに、ＡからＩには、すでに数字が与えられている場合もある.

```
      2
    B C
+ D 8 F
-------
  G 3 I
```

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

出題のねらい

- 桁が別々に与えられたときの、３桁の足し算の結果の計算が必要.
- 欠けている数字を、どう補うか考える必要がある.
- 可能性のある配置をどのように試すか、が問われている.
- 様々なアプローチ.

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

解法１

- 再帰関数で、可能な数字の配置をすべて試す.
- すでに使用した数字を保持しながら、順番にマスを埋めていく.

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

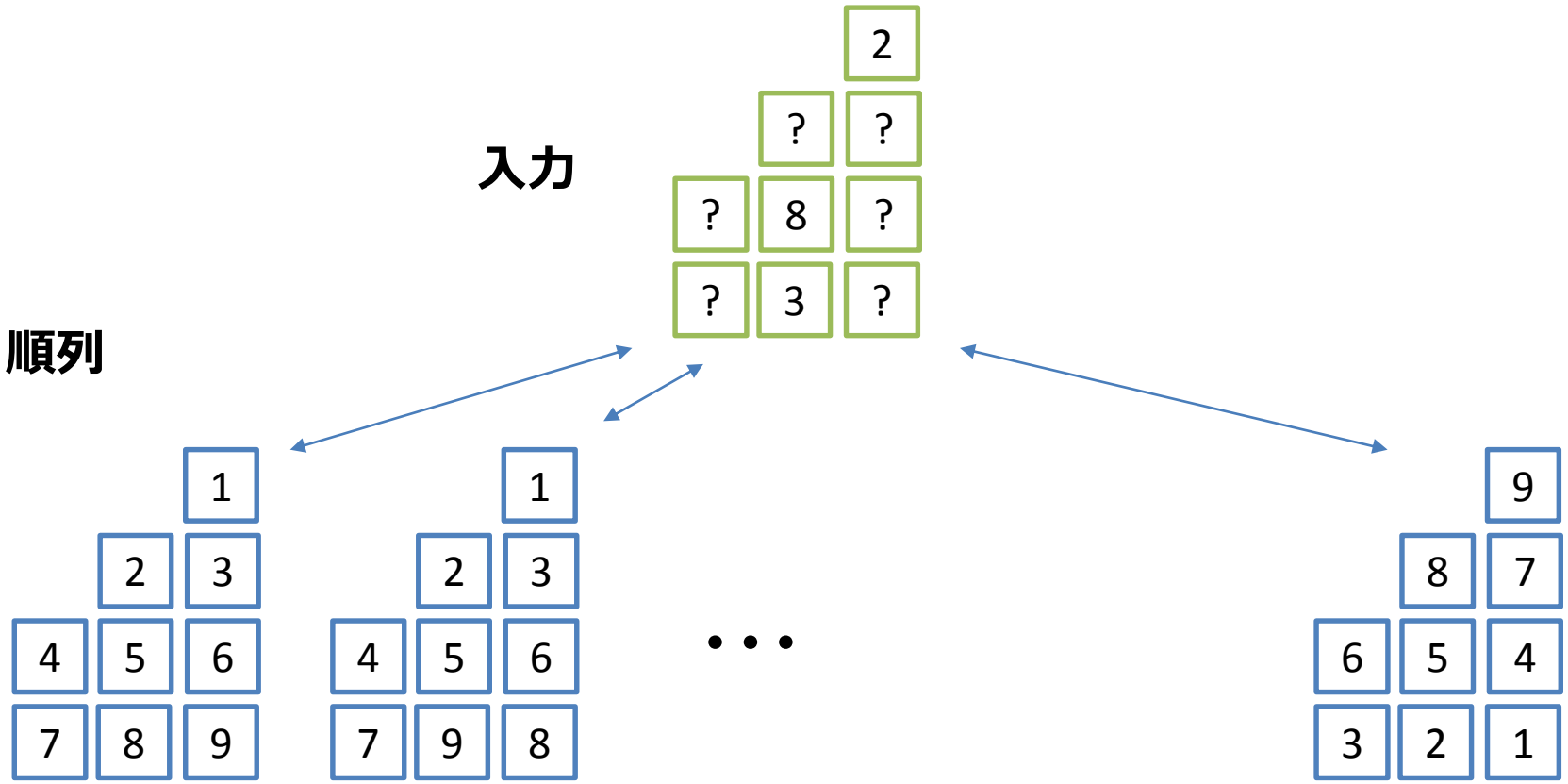
解答例（Ｃ＋＋）

```
int num[10],used[10],ans;
int cnt;

void saiki(int kai){
  cnt++;
  if(kai == 9){
    int up = 100*num[3]+10*(num[1]+num[4])+(num[0]+num[2]+num[5]);
    int down = 100*num[6]+10*num[7]+num[8];
    if(up == down ) ans++;
  }
  else if(num[kai] != -1) saiki(kai+1);
  else
    for(int i=1;i<=9;i++){
      if(used[i] == 1) continue;
      used[i] = 1, num[kai] = i;
      saiki(kai+1);
      used[i] = 0, num[kai] = -1;
    }
}

int main() {
  for(int i=0;i<9;i++)cin >> num[i], used[max(0,num[i])] = 1;
  saiki(0);
  cout << ans << endl;
  return 0;
}
```

問３　虫食い算

解法２

- 1, 2, … 9 の順列を全て作る　→　next_permutation.
- 入力の配置にマッチし、かつ計算結果が一致する順列について、カウント＋１.



問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

解答例（Ｃ＋＋）

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

int main() {
  int a[9],b[9];
  for(int i=0; i<9; i++) {
    a[i]=i+1;
    cin >> b[i];
  }
  int ans=0;
  do {
    bool f=1;
    for(int i=0; i<9; i++) {
      if(b[i]!=-1 && a[i]!=b[i]) f=0;
    }
    if(!f) continue;
    int d[4];
    memset(d,0,sizeof(d));
    d[0]=a[0];
    d[1]=a[1]*10+a[2];
    d[2]=a[3]*100+a[4]*10+a[5];
    d[3]=a[6]*100+a[7]*10+a[8];
    if(d[0]+d[1]+d[2]==d[3]) ans++;
  } while(next_permutation(a,a+9));
  cout << ans << endl;
  return 0;
}
```

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

解法3

- 欠けている数字が何か記憶して、その順列を全てつくる.
- 作った各順列を、欠けている桁に埋めてみた結果、演算が正しければ、カウント＋１.

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

解答例（Ｃ＋＋）

```
#define ALL(c)  (c).begin(),(c).end()
typedef std::vector<int>    vInt;

int SumRes( const vInt& digit ) {
  return digit[6]*100 + digit[7]*10 + digit[8];
}

int SumCalc( const vInt& digit ) {
  return                              digit[0]
                  + digit[1]*10 + digit[2]
    + digit[3]*100 + digit[4]*10 + digit[5];
}

bool IsValidSum( const vInt& digit ) {
  return SumRes(digit) == SumCalc(digit);
}
```

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

解答例（Ｃ＋＋）

```
int Count( vInt& empty, const vInt& input ) {
  if ( empty.empty() && IsValidSum( input ) ) return 1;
  int n = 0;
  do {
    if ( IsValidSum( Fill( empty, input ) ) ) ++n;
  } while ( std::next_permutation( ALL(empty) ) );
  return n;
}

vInt Fill( const vInt& empty, const vInt& input ) {
  vInt res( input );
  for ( size_t i=0, j=0; i<input.size(); ++i ) {
    if ( input[i] == -1 ) res[i] = empty[j++];
  }
  return res;
}
```

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

解答例（Ｃ＋＋）

```
int main() {
  vInt input(9);
  vInt tmp(9);
  vInt empty;
  bool exist[9];
  if ( scanf("%d %d %d %d %d %d %d %d %d",
              &input[0], &input[1], &input[2],
              &input[3], &input[4], &input[5],
              &input[6], &input[7], &input[8]) != 9 ) return 0;

  std::fill( exist, exist+9, false);
  for ( int i=0; i<9; ++i ) {
    if ( input[i] > 0 ) exist[input[i]-1] = true;
  }

  //欠けている数字のリスト
  for ( int i=0; i<9; ++i ) {
    if ( !exist[i] ) empty.push_back( i+1 );
  }
  printf("%d¥n", Count( empty, input ));
  return 0;
}
```

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

解法4

- ９重ループ（ループ回数$9^9=4x10^7$程度）で全部試す、気迫のループ解も可能（非推奨）.

問３　虫食い算

AIZU
THE UNIVERSITY OF AIZU

解答例（非推奨）

```
#define rep(i, n) for ( int i = 1; i <= n; i++ )
void solve(){
  int X[10], A, B, C, D, E, F, G, H, I;
  bool U[10];
  rep(i, 9) cin >> X[i];
  rep(i, 9) U[i] = false;
  int sum = 0;
  rep(a,9) {
    U[a] = true;
    rep(b,9) {
      if (U[b]) continue;
      U[b] = true;
      rep(c,9) {
        if (U[c]) continue;
        U[c] = true;
        rep(d,9) {
          if (U[d]) continue;
          U[d] = true;
          rep(e,9) {
            if (U[e]) continue;
            U[e] = true;
            rep(f,9) {
              if (U[f]) continue;
              U[f] = true;
              rep(g,9) {
                if (U[g]) continue;
                U[g] = true;
                rep(h,9) {
                  if (U[h]) continue;
                  U[h] = true;
                  rep(i,9){
                    if (U[i]) continue;

                      省略

                      if ( I != (A + C + F)%10 ) continue;
                      if ( H != ((A + C + F)/10 + B + E)%10 ) continue;
                      if ( G != ((A + C + F)/10 + B + E)/10 + D ) continue;
                      sum++;
                    }
                  U[h] = false;
                }
                U[g] = false;
              }
              U[f] = false;
            }
            U[e] = false;
          }
          U[d] = false;
        }
        U[c] = false;
        }
      U[b] = false;
    }
    U[a] = false;
 }
  cout << sum << endl;
}
```

問４　貴金属リサイクル

AIZU
THE UNIVERSITY OF AIZU

問題概要

- 重さが「ボッコ」で、数が「マルグ」という単位の「アイヅニウム」という金属の塊がいくつか与えられる.
- 与えられたアイヅニウムを、炉の中に入れて溶かし、いくつかのアイヅニウムの塊を再生する.
- このとき、なるべくアイヅニウムの塊の数が少なくなるようにする.

- xボッコは２のｘ乗グラムで、yマルグは２のｙ乗個で、入力としてxとyの組みが与えられる.
  - ｘ = 2ボッコ ⇔ 2^2=4グラム、ｙ = 1マルグ ⇔ 2^1=2個
  - ｘ = 0ボッコ ⇔ 2^0=1グラム、ｙ = 0マルグ ⇔ 2^0=1個
- 入力も出力も、ボッコ単位、マルグ単位.

問４　貴金属リサイクル

AIZU
THE UNIVERSITY OF AIZU

出題のねらい

- 最大10万ボッコ＝ 2^{100000}なので、グラムに直すと３万桁.
- マルグも同様.
- グラムに直して、多倍長整数で処理すると、一番単位の大きな１０万ボッコが10万マルグ与えられただけで３万以上＊３万以上＝１０^{9}程度のオーダで間に合わない.
- 数学的な考察が必要.

$$2^a \times 2^b = 2^{a+b}$$

問４　貴金属リサイクル

AIZU
THE UNIVERSITY OF AIZU

解法

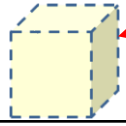
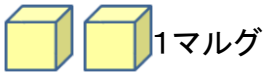
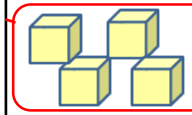
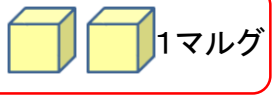
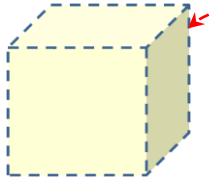
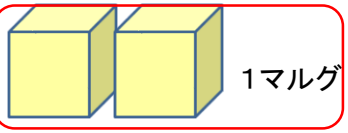
- ボッコもマルグも、基数が２の指数なので、指数の足し算で済む.
- ただし、下の桁からの繰り上がりを考える必要がある.
- ０ボッコや０マルグというのは、０グラムや０個でないことに注意.

問４　貴金属リサイクル

AIZU
THE UNIVERSITY OF AIZU

例

x_1 ＝ 2ボッコ（2^2=4グラム）、y_1 ＝ 1マルグ（2^1=2個）
x_2 ＝ 1ボッコ（2^1=2グラム）、y_2 ＝ 2マルグ（2^2=4個）

	4ボッコ(16g)	3ボッコ(8g)	2ボッコ(4g)	1ボッコ(2g)	0ボッコ(1g)
1ステップ目					
2ステップ目					
3ステップ目					

結果：　４ボッコが０マルグ

問４　貴金属リサイクル

AIZU
THE UNIVERSITY OF AIZU

解答例（Ｃ＋＋）

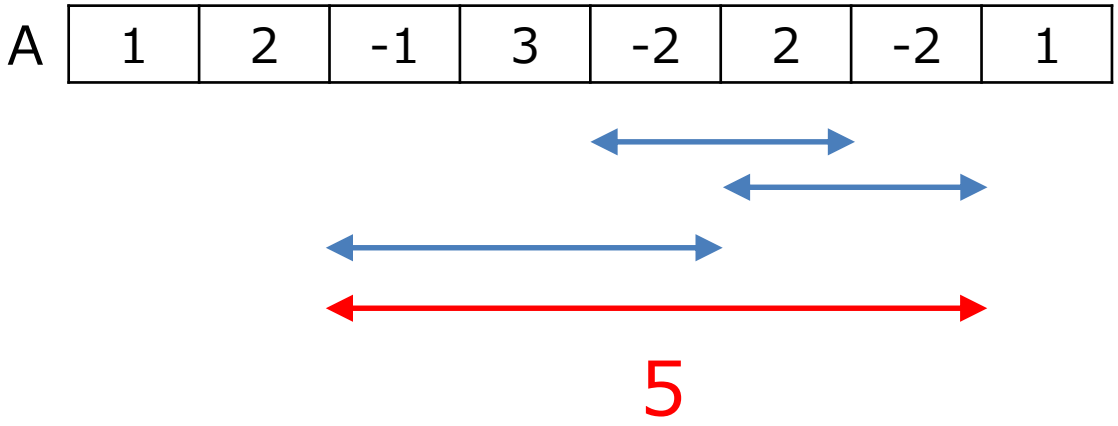
```
static const int PMAX = 100000;

int N, T[2*PMAX+20+1];

int main(){
  int a, b;
  for ( int i = 0; i <= 2*PMAX+20; i++ ) T[i] = 0;
  cin >> N;
  for ( int i = 0; i < N; i++ ){
    cin >> a >> b;
    T[a+b]++;
  }
  for ( int i = 0; i < 2*PMAX+20; i++ ){
    T[i+1] += T[i]/2;
    T[i] = T[i]%2;
    if ( T[i] ) cout << i << " " << 0 << endl;
  }

  return 0;
}
```

問５　完全平等二国間貿易

AIZU
THE UNIVERSITY OF AIZU

問題概要

- 数列 A = a_1, a_2, …, a_N の連続する区間で、その和が0となる最長の区間の長さを求めよ.
- $1 \leqq N \leqq 200000$, $-1 \times 10^9 \leqq a_i \leqq 10^9$

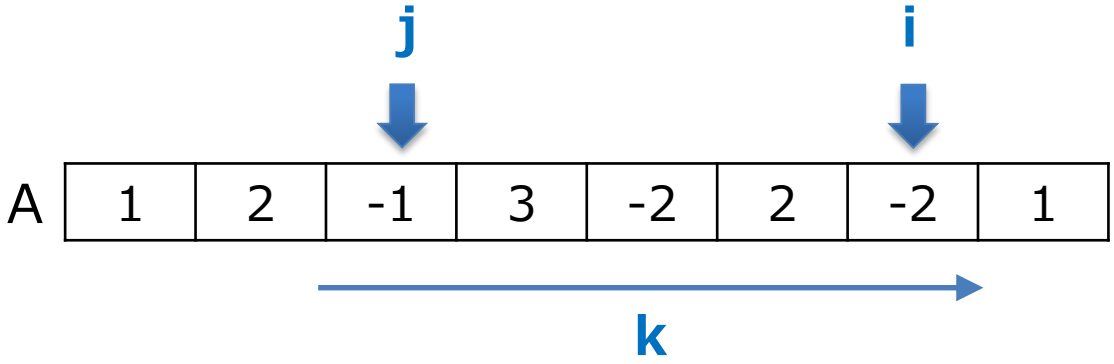


出題のねらい

- 数列の長さＮが大きいため、効率の良いアルゴリズムが必要.
- 計算量の見積もりと、高度なデータ構造の応用ができるか.

問５　完全平等二国間貿易

力任せのアルゴリズム①

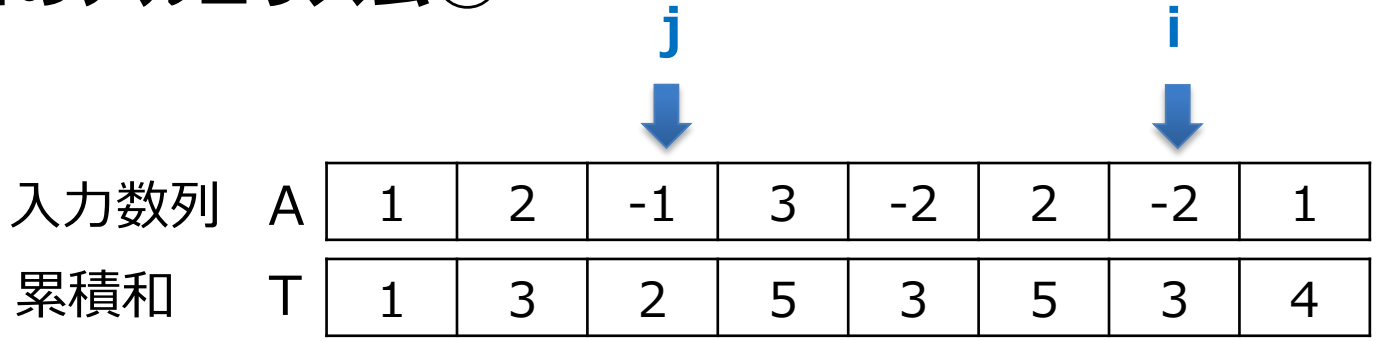



```
for ( int i = 1; i <= N; i++ ){
  for ( int j = 1; j <= i; j++ ){
    long long sum = 0;
    for ( int a = j; a <= i; a++ ) {
      sum += A[a];
    }
    if ( sum == 0 ) ans = max(i-j+1, ans);
  }
}
```

- 両端の位置 (i, j) を決め打ちして、その範囲内の値を加算する.
- i, j, k の３重ループで $O(N^3)$. 時間制限

問５　完全平等二国間貿易

力任せのアルゴリズム②




```
for ( int i = 1; i <= N; i++ ) {
  cin >> A[i];
  T[i] = T[i-1] + A[i];
}

for ( int i = 1; i <= N; i++ ){
  for ( int j = 0; j < i; j++ ){
    if ( T[i] - T[j] == 0 ) ans = max(i - j, ans);
  }
}
```

- 両端の位置 (i, j) を決め打ちして、累積和を利用して総和を計算する.
- i, j の２重ループで $O(N^2)$. 時間制限

問５　完全平等二国間貿易

AIZU
THE UNIVERSITY OF AIZU

解法

- 累積和を順番にたどる．現在地より前にある同じ値の累積和を探す．
- 複数ある場合は、最長になるように、最も前方にあるものを選ぶ．
- 連想配列を用いて、累積和 s が出現した最初の位置を記録しておく．

問５　完全平等二国間貿易

AIZU
THE UNIVERSITY OF AIZU

解法

```
map<long long, int> M;
long long sum = 0;

for ( int i = 1; i <= N; i++ ) {
  cin >> a;
  sum += a;
  if ( M.find(sum) == M.end() ){
    M[sum] = i;
  } else {
    int j = M[sum];
    ans = max(ans, i-j);
  }
}
```

- 連想配列から累積和 s を探す操作をＮ回行って、O(N log N).
- STL の map （C++）や TreeMap(Java) を利用することができる.

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

問題概要

- TinyPowerというプログラミング言語を、解釈しシミュレーションする.

文の種類	動作
ADD var_1 var_2 var_3	変数var_2の値とvar_3の値を加算した結果を変数var_1に代入する
ADD var_1 var_2 con	変数var_2の値と定数conを加算した結果を変数var_1に代入する
SUB var_1 var_2 var_3	変数var_2の値からvar_3の値を減算した結果を変数var_1に代入する
SUB var_1 var_2 con	変数var_2の値から定数conを減算した結果を変数var_1に代入する
SET var_1 var_2	変数var_2の値を変数var_1に代入する
SET var_1 con	定数conを変数var_1に代入する
IF var_1 dest	変数var_1の値が０でないときだけ、行番号destにジャンプする
HALT	プログラムを停止させる

- 変数の数は５個まで.

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

問題概要

- TinyPowerは、以下の状況でのみ、停止する.
  - HALT文を実行したとき.
  - 負の整数または１６以上の整数を変数に代入しようとしたとき（変数の値は更新されない）.
  - プログラムに現れない行番号にジャンプしようとしたとき.
  - プログラムの最後の文を実行した後、そこからどの行にもジャンプしないとき.
- プログラムが停止するかしないか判定し、停止する場合は、初めて停止する時点での変数の値を全て出力する.
- 制限時間１０秒.

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

出題のねらい

- 停止する場合は、停止の条件を満たすときなので、シミュレーションしていって判定すればよい.
- 停止しない場合は、どうやって判定するかがやや難しい.

- 実装力が問われる問題.
  - 文字列の操作（比較、数値への変換など）.

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

解法１

- 文を解釈しながら、変数の値を更新するシミュレーション.
- 停止しないのは、同じ状態が現れるとき（同じ状態が現れたらそこから先の実行は前と同じになるため）.
- 状態は、全ての変数の値と現在実行中の文の行番号の組.
  - １行実行するたびに、現在の状態を整数にエンコードしておいて、過去に同じ整数が現れたか判定すれば良い.
  - または、visited[行][v_1][v_2][v_3][v_4][v_5]の状態空間を訪問.

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

解答例の一部（Ｊａｖａ）　　プログラムの文の解釈

```
static Op Interpret( int lineNum, int type, String[] args ) {

        Op res = new Op();
        res.m_line = lineNum;
        res.m_instr = type;

        for ( int i=0; i<3; ++i ) {
            if ( args[i] == null ) break;
            if ( Character.isLowerCase(args[i].charAt(0)) ) {

                  res.m_opt[i] = DataType.CHAR;

                  if ( !m_mapVar.containsKey( args[i].charAt(0) ) ) {
                      res.m_opr[i] = m_nVar;
                      m_var[m_nVar] = new Variable(); //新しい名前の変数
                      m_var[m_nVar].name = args[i].charAt(0);
                      m_mapVar.put( args[i].charAt(0), m_nVar++ );
                  }
                  else  res.m_opr[i] = m_mapVar.get( args[i].charAt(0) ); //もう現れた変数
            }
            else {
                  res.m_opt[i] = DataType.INT;
                  res.m_opr[i] = Integer.parseInt(args[i]); //整数のときは即値
            }
        }

        return res;
}
```

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

解答例の一部（Ｊａｖａ）　代入(SET)、加算(ADD)命令等の実装

```
static boolean IsValidVar( int v ) { return v >= 0 && v <= 15; }
static int Var( int i ) { return m_var[i].value; }

static boolean SetI( int i, int v ) { //set immediate
      if ( !IsValidVar( v ) ) return false;
      m_var[i].value = v;
      return true;
}

static boolean SetV( int i, int j ) { return SetI( i, m_var[j].value ); }

static boolean AddI( int i, int j, int v ) {
      return SetI( i, m_var[j].value + v );
}
static boolean AddV( int i, int j, int k ) {
      return SetI( i, m_var[j].value + m_var[k].value );
}

static boolean SubI( int i, int j, int v ) {
      return SetI( i, m_var[j].value - v );
}
static boolean SubV( int i, int j, int k ) {
      return SetI( i, m_var[j].value - m_var[k].value );
}
```

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

解答例の一部（Ｊａｖａ）　状態の整数エンコード

```
static int Encode( int line ) {
      int res = 0;
      for ( int i=0; i<m_nVar; ++i ) res |= (m_var[i].value) << (4*i);
      return res | (line << (4*m_nVar));
}
```

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

解答例（Ｊａｖａ）

シミュレーション部分

```
boolean state[] = new boolean[1<<26];
for ( int i=0; i<1<<26; ++i ) state[i] = false;

for ( int i=0; ; ) {
    boolean branch = false;
    boolean valid = true;
    Op op = program.get( i );
    switch ( op.m_instr ) {
    case 0:
          if ( op.m_opt[1] == Op.DataType.CHAR )
              valid = Op.SetV( op.m_opr[0], op.m_opr[1] );
          else
              valid = Op.SetI( op.m_opr[0], op.m_opr[1] );
          break;
…中略
    if ( !branch ) {
          ++i;
          int s = Op.Encode(i);
          if ( state[s] ) {//既に通った状態をまた通った
              System.out.println("inf");
              return;
          }
          state[s] = true;
    }
```

問６　プログラム停止判定

AIZU
THE UNIVERSITY OF AIZU

解法2

- 文を解釈しながら、変数の値を更新するシミュレーション.
  - 全状態数（50行×(0 から15の16通り)$^{5\text{変数}}$＝52,428,800)回シミュレーションしても終わらなければ「inf」にする.
  - 状態を保持しなくてよいので実装が楽になる.

問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

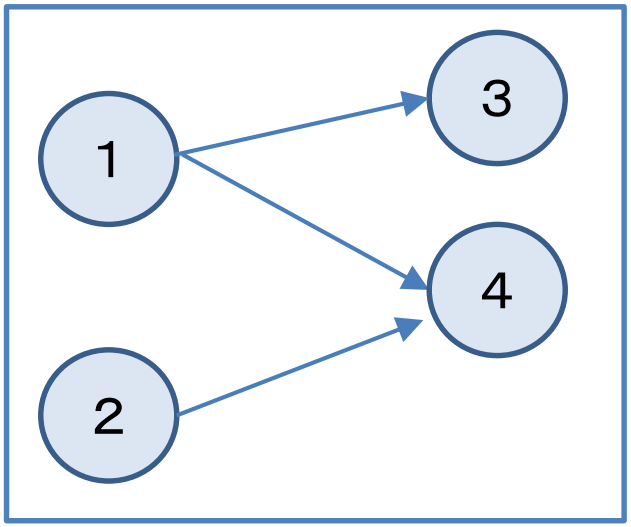
問題概要

- 依存関係を持つタスクがいくつか与えられたとき、それらの実行順序を報告せよ.
- 現時点で複数のタスクが実行可能な場合、タスクが持つ複数の属性値を比較して、どのタスクを先に実行するか決定する.
- ただし、属性値の比較順が途中で変更される.

問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

例



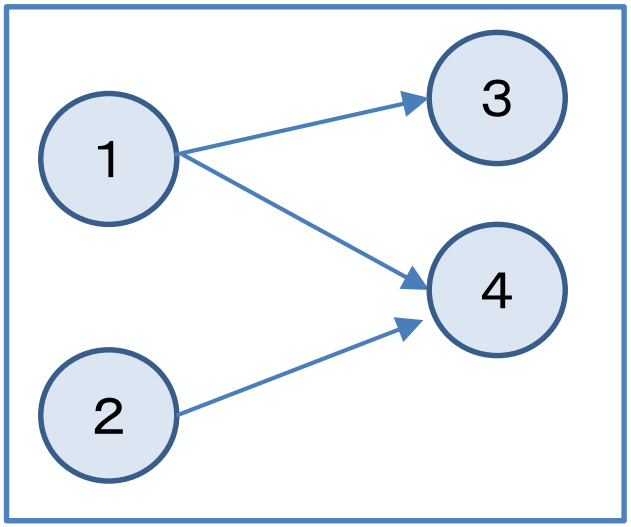
依存関係：

- １と２は、何にも依存していないので、初めから実行できる.
- ３は、１に依存しているので、１の実行が終わったら実行できる.
- ４は、１と２に依存しているので、１と２の実行が終わったら実行できる.

問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

例



例：
最初の評価順：属性１→属性２
２つ処理を終えた時点で
属性２→属性１に変更する

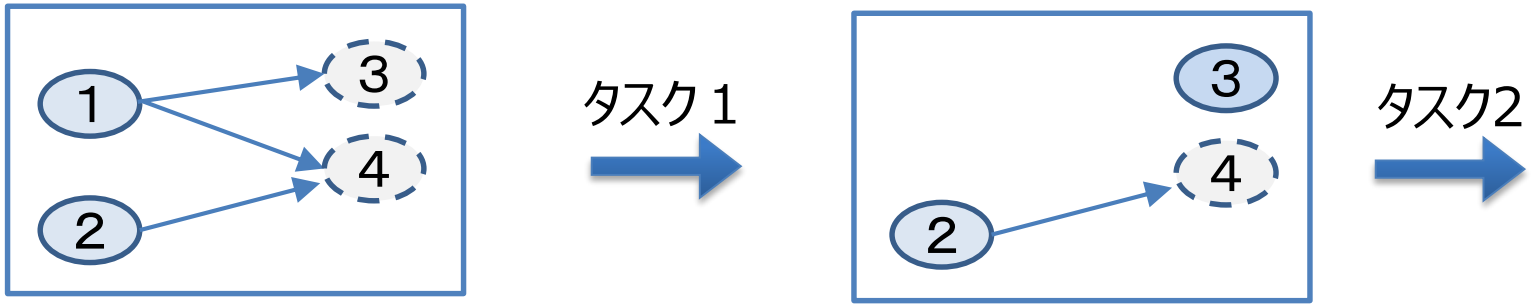
	属性1	属性2
タスク1	4	1
タスク2	3	2
タスク3	2	3
タスク4	1	4

問７　スケジューラ

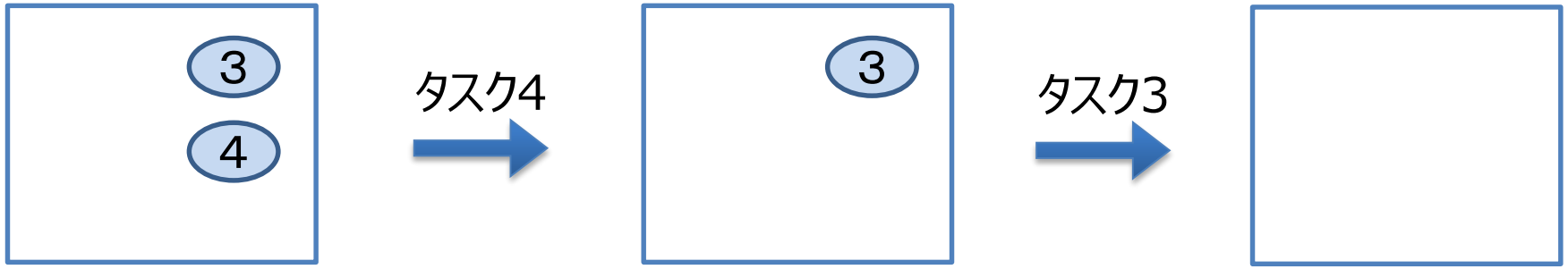
AIZU
THE UNIVERSITY OF AIZU

例

評価順：属性１→属性２



評価順：属性２→属性１



問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

出題のねらい

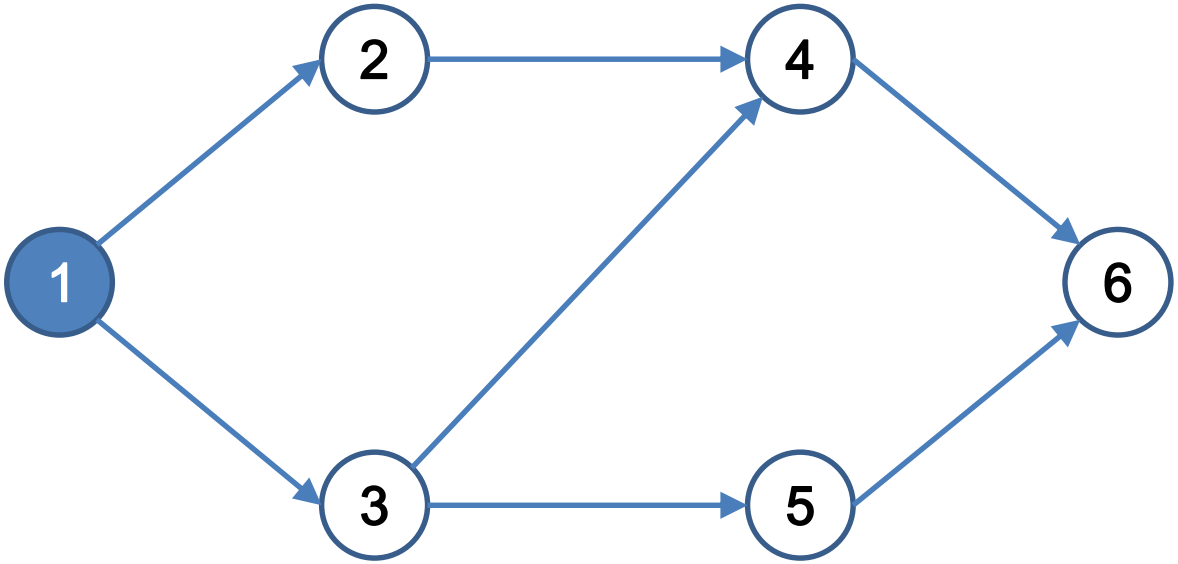
- タスクの数Nは最大50000までなので、現在実行可能なタスク全てを、現在の優先順位に従って比較すると時間切れ (O(N^2)).
- 依存関係を解決しながら、現在の優先順位でどのタスクが選ばれるか効率よく求める必要がある.
- 実装力が問われる.

問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

トポロジカルソート：キュー

- 入次数が０のノード（タスク）をキューにいれていく.
- キューの先頭から削除（タスクを処理）.

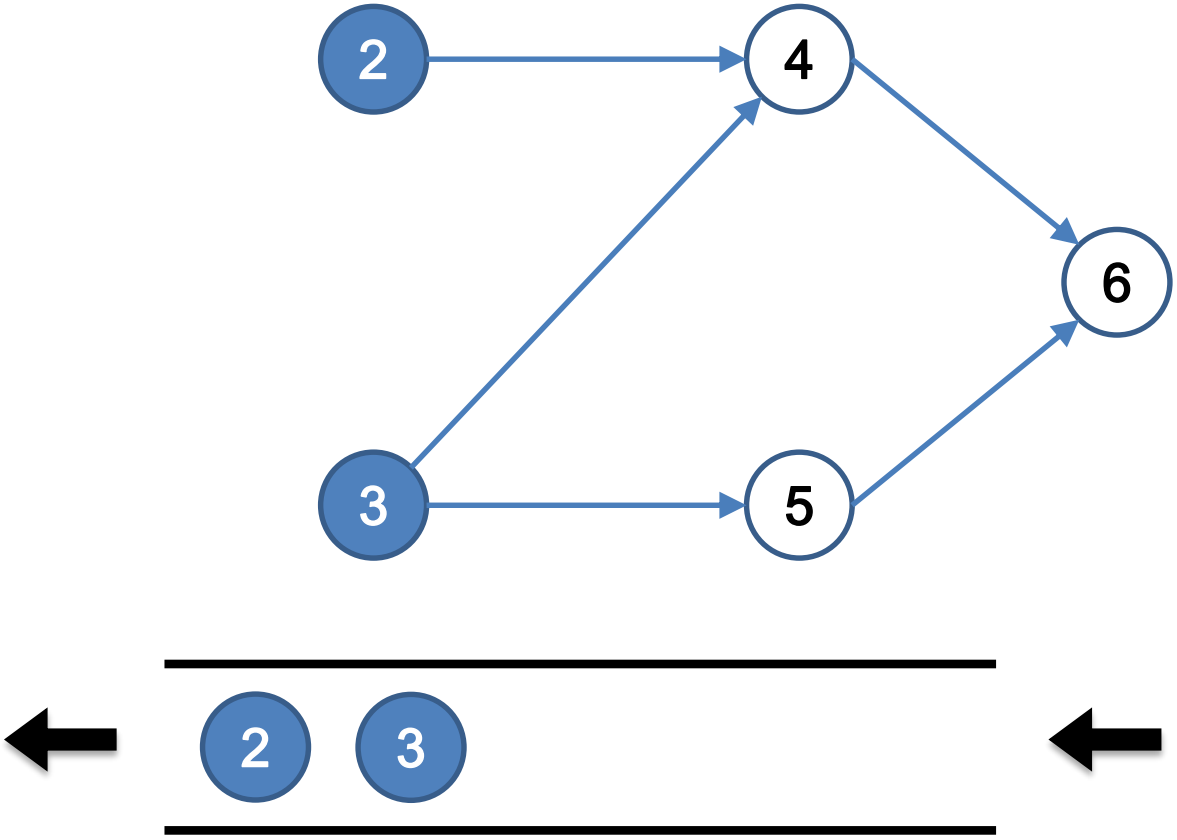


問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

トポロジカルソート：キュー

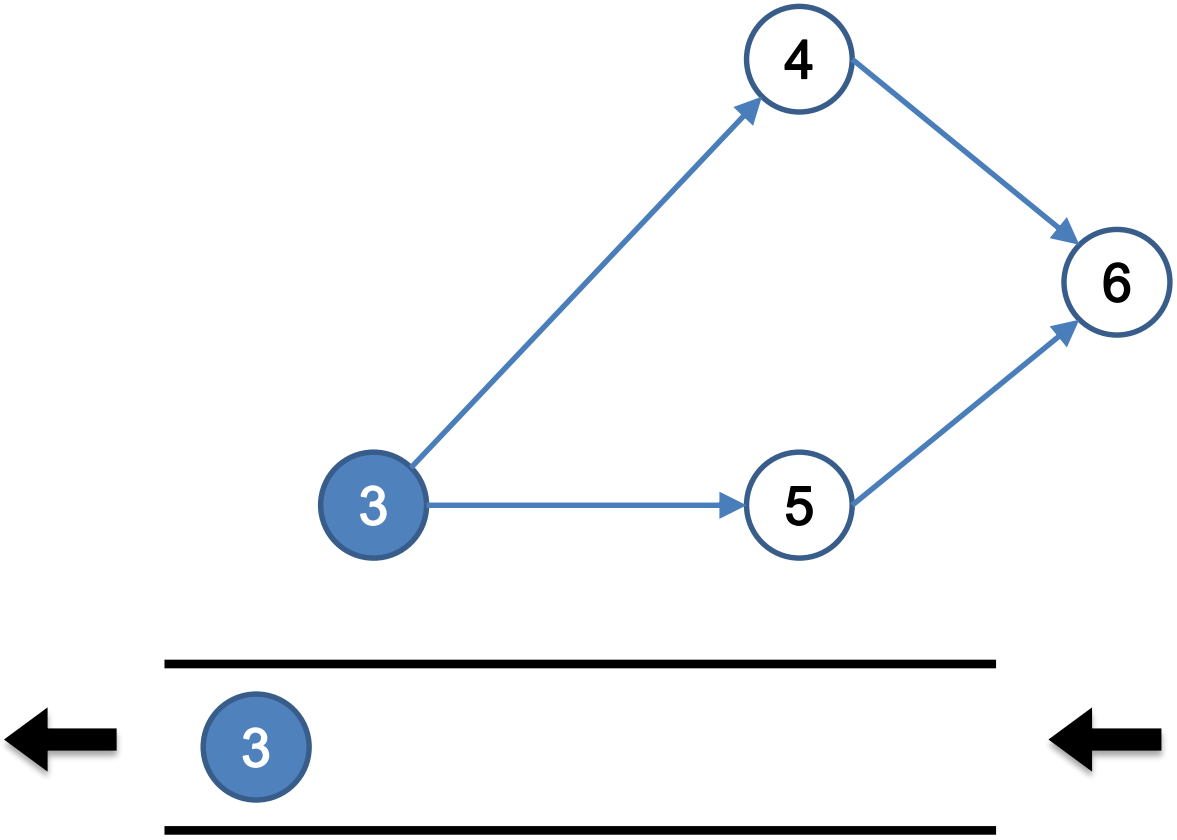
- 入次数が０のノード（タスク）をキューにいれていく.
- キューの先頭から削除（タスクを処理）.



問７　スケジューラ

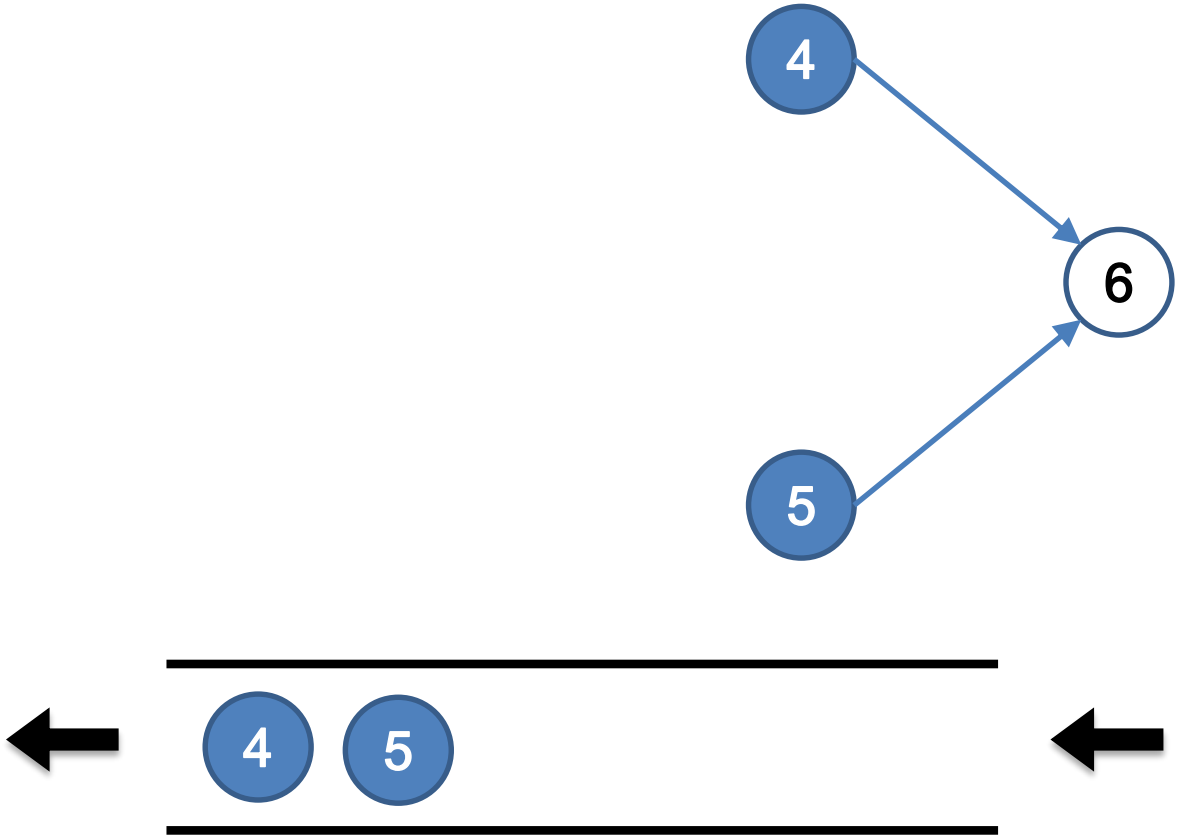
AIZU
THE UNIVERSITY OF AIZU

トポロジカルソート：キュー



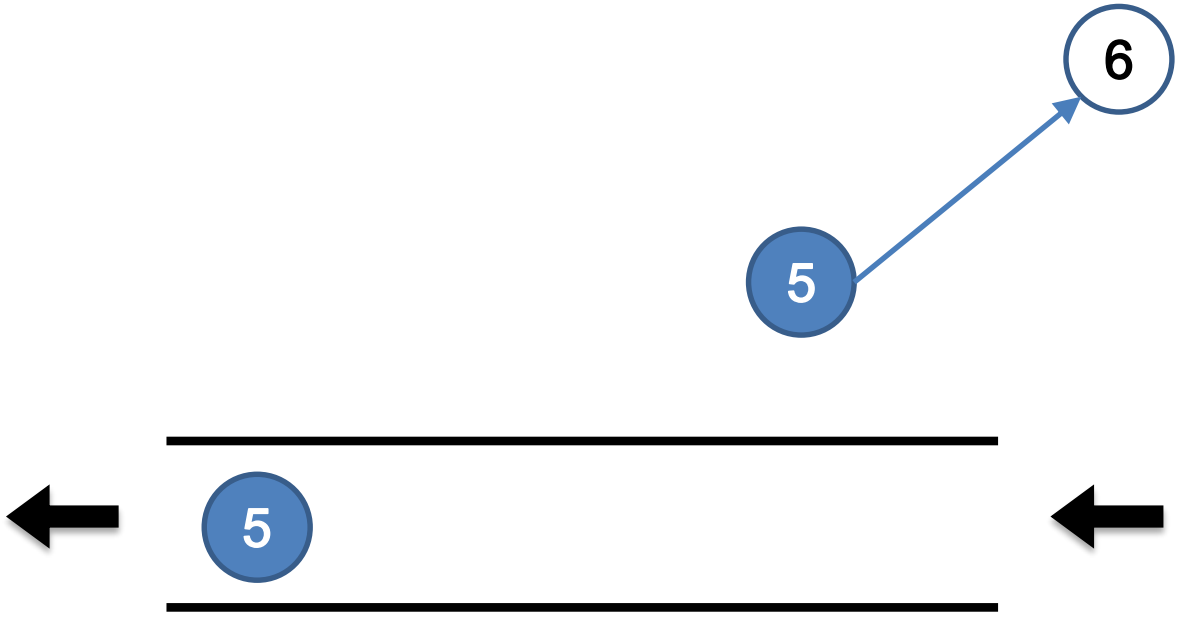
問７　スケジューラ

トポロジカルソート：キュー



問７　スケジューラ

トポロジカルソート：キュー



問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

トポロジカルソート：キュー

問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

解法

- キューによるトポロジカルソート → 優先度付きキューによるトポロジカルソート.
- すべての評価順序を考慮して、K!個のキューを準備する.
- O(N × K! × log N).
- 優先度付きキューの実装
  - ヒープを実装
  - priority_queue
  - set

問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

解法：実装の注意点

- 属性データをすべてキューに入れてしまう方法は厳しい.
  - 例えば　priority_queue<vector<int>> PQ[K!];
- キューにはノード番号のみを入れ、比較の際に属性値の表を参照するようにする.

問７　スケジューラ

AIZU
THE UNIVERSITY OF AIZU

解答例（抜粋）

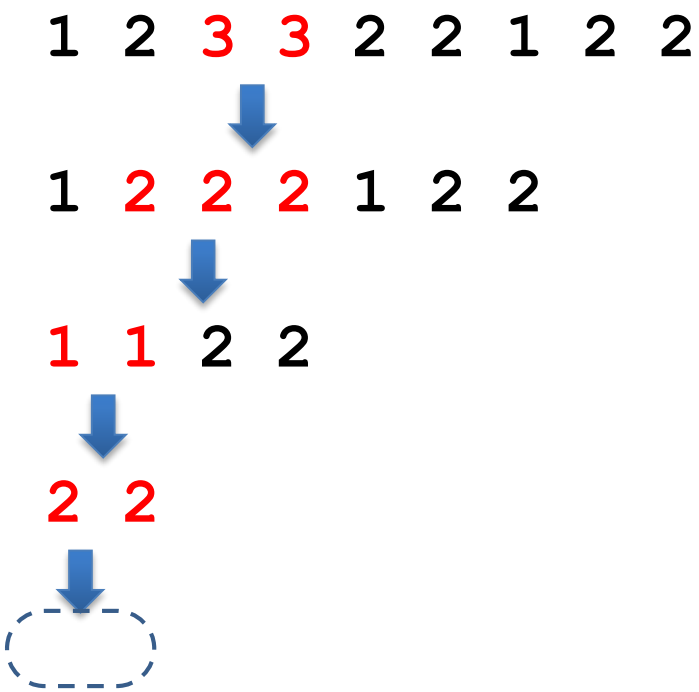
```
class Task{
public:
  int v;
  int pid;
  Task(int v=0, int pid=0):v(v), pid(pid){}

  bool operator < ( const Task &t) const{
    for ( int i = 0; i < K; i++ ){
      if ( F[v][T[pid][i]] == F[t.v][T[pid][i]] ) continue;
      return F[v][T[pid][i]] < F[t.v][T[pid][i]];
    }
  }
};

priority_queue<Task> PQ[SMAX];
```

問８　消える数列、消えない数列

AIZU THE UNIVERSITY OF AIZU

問題概要

- 1から9の数字からなる、列が与えられる.
- 同じ数が連続するグループを、まとめて、消すことができる.
- 与えられた数列が消せるか消せないか判定せよ.



問８　消える数列、消えない数列

AIZU
THE UNIVERSITY OF AIZU

考察

- 数列の長さは１以上100以下.
  → 探索で調べるのは厳しい
- 数字は１から９のみ.
  → 状態が持てそう

問８　消える数列、消えない数列

AIZU THE UNIVERSITY OF AIZU

解法：区間DP　（動的計画法）

- dp[i][j][0] が true なら、区間[i,..,j]は消せる.
- dp[i][j][c] が true なら、区間[i,..,j]に c（だけ）を残せる.

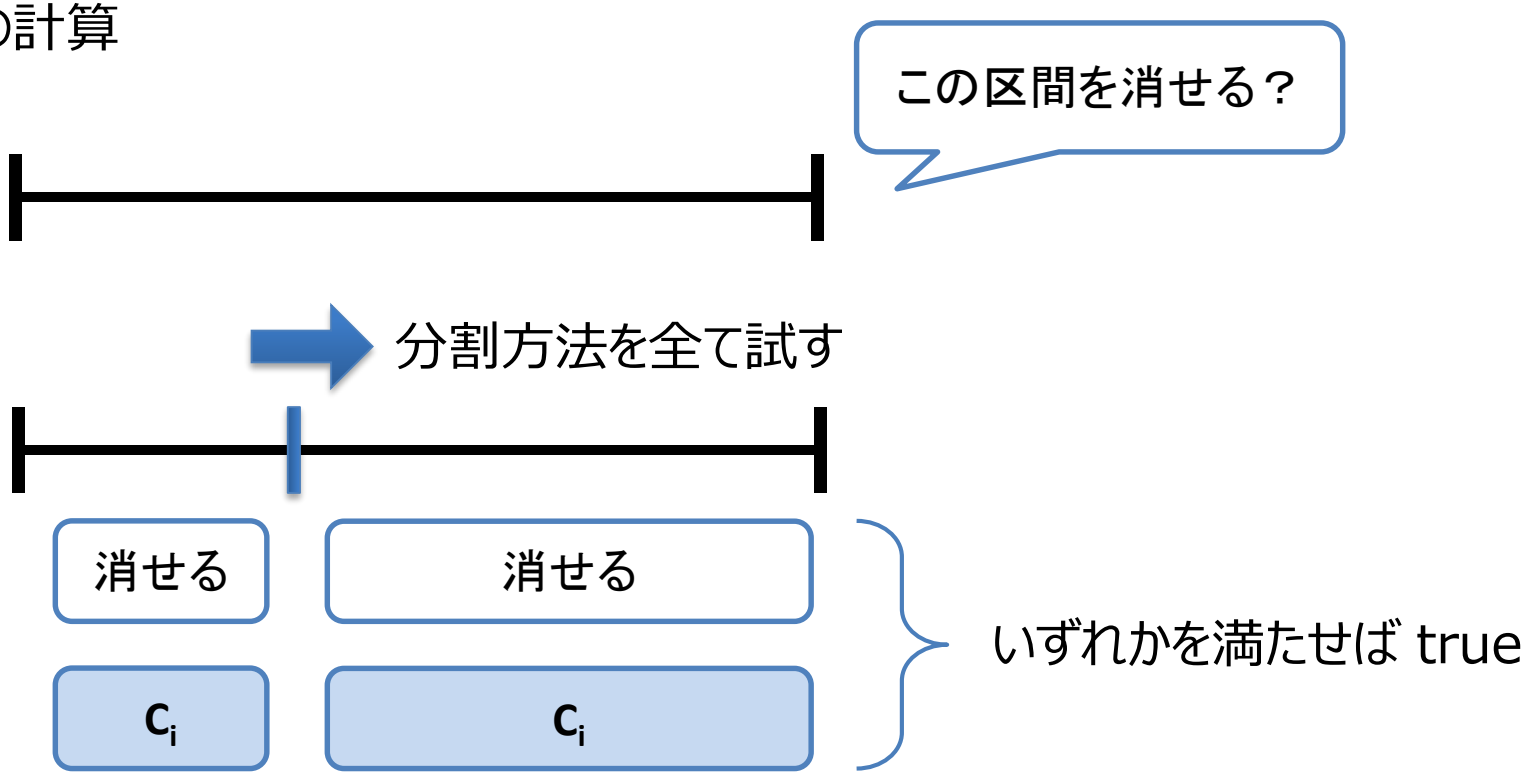
- N × N × N × K の３重ループ、または
- N × N × K 空間のメモ化再帰.
- $O(K \times N^3)$.

問８　消える数列、消えない数列

AIZU THE UNIVERSITY OF AIZU

解法：区間DP　（動的計画法）

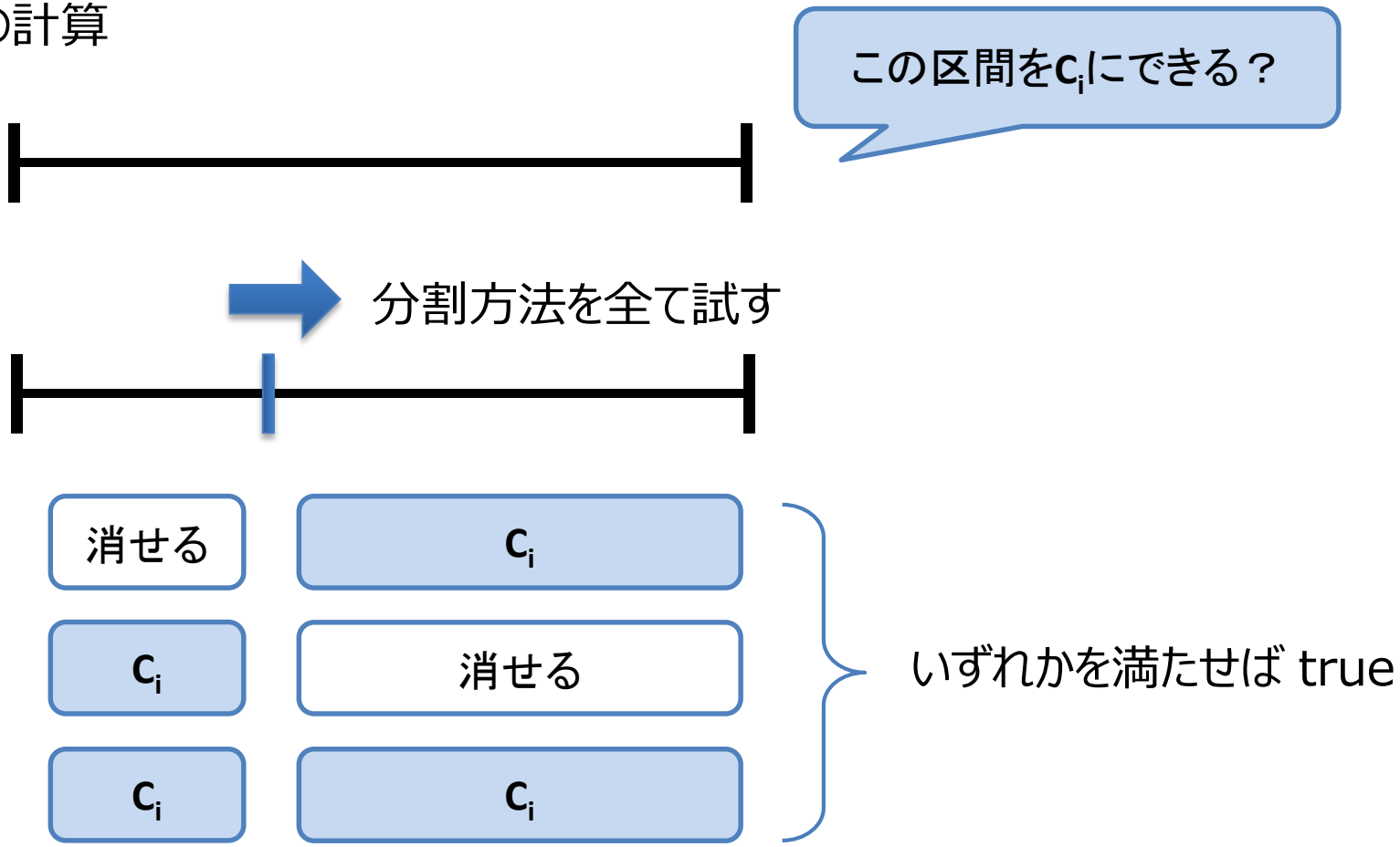
- 区間の計算



問８　消える数列、消えない数列

解法

- 区間の計算



問８　消える数列、消えない数列

AIZU THE UNIVERSITY OF AIZU

解法

- 区間の計算

```
for ( int l = 2; l <= N; l++ ){
   for ( int i = 0; i <= N-l; i++ ){
     int j = i+l-1;
     for ( int k = i; k <= j-1; k++ ){
       for ( int c = 0; c <= 9; c++ ){
         if ( dp[i][k][0] && dp[k+1][j][0] ||
              dp[i][k][c] && dp[k+1][j][c] ) dp[i][j][0] = true;
         if ( c ){
           if ( dp[i][k][0] && dp[k+1][j][c] ||
                dp[i][k][c] && dp[k+1][j][0] ||
                dp[i][k][c] && dp[k+1][j][c] ) dp[i][j][c] = true;
         }
       }
     }
   }
 }
```

問８　消える数列、消えない数列

AIZU
THE UNIVERSITY OF AIZU

参考

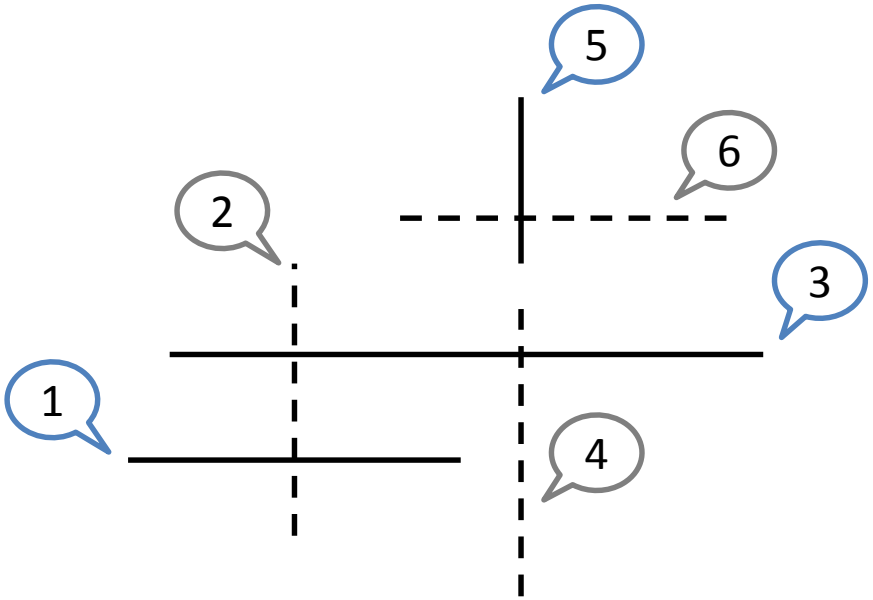
- CYKアルゴリズム.
  - 文法を与えて、その文法に属する文字列かどうかを判定するアルゴリズム.

問９　線分配置

AIZU
THE UNIVERSITY OF AIZU

問題概要

- 順番に与えられる、軸に平行な線分を、平面上に配置する.
- ただし、既に配置されている線分と触れる場合は配置しない.
- 各線分が、配置できたかどうかを判定せよ.
- 線分の数Ｎ≦100,000.　座標の最大値は 10^9.



問９　線分配置

AIZU
THE UNIVERSITY OF AIZU

全探索

- 線分が与えられる度に、これまでに配置されているどの線分とも触れないかどうか、チェックする.
- $O(N^2)$. 時間制限.

問９　線分配置

AIZU
THE UNIVERSITY OF AIZU

バケット

- 平面をＲ×Ｃの区画に分割して、該当する区画のみ調べる.
  - ある区画に集中するような入力に弱い.
- $O(N^2)$. 時間制限.

問９　線分配置

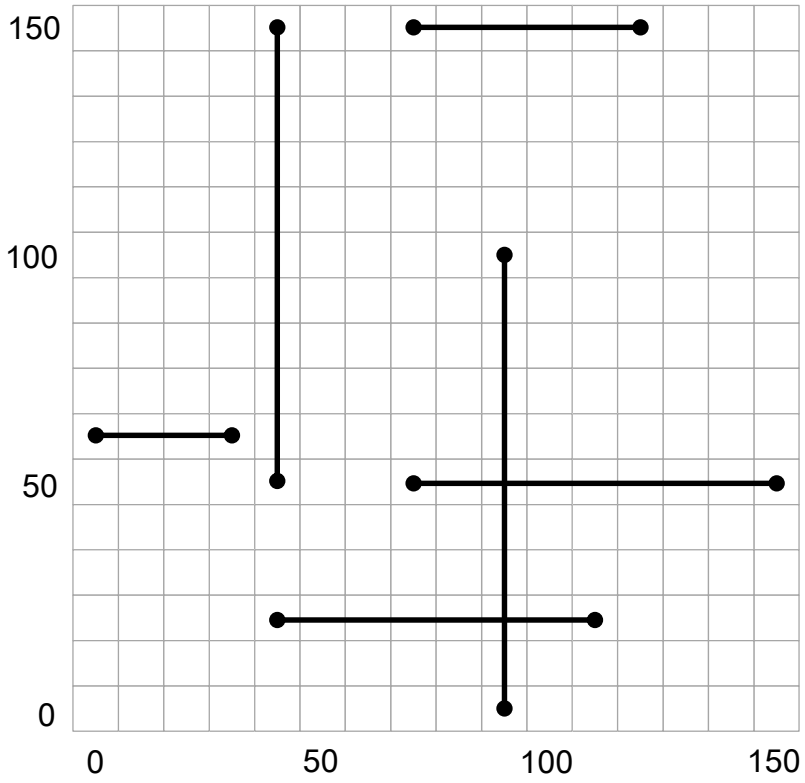
AIZU
THE UNIVERSITY OF AIZU

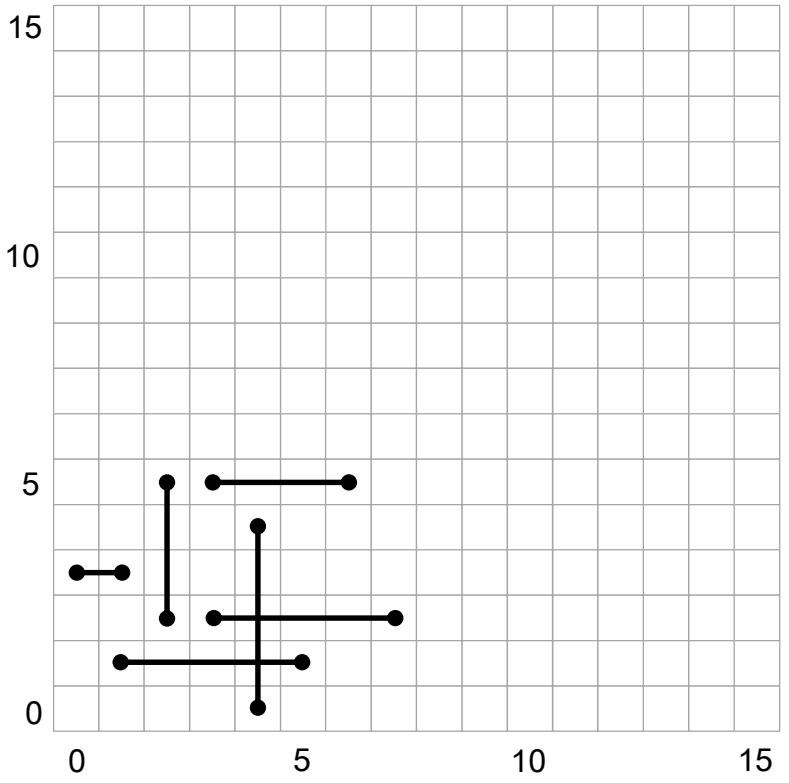
解法：座標圧縮＋平衡二分木+セグメントツリー

- 線分の位置関係のみを保持すればよい（正確な座標は必要ない）.
  → 座標圧縮
- 集合に対する要素の追加・検索を効率的に行いたい.
  → 平衡二分木　（set等）
- 区間における要素の追加・検索を効率的に行いたい.
  → セグメントツリー

問９　線分配置

座標圧縮





最大1,000,000,000

最大2 × 100,000

問９　線分配置

AIZU
THE UNIVERSITY OF AIZU

線分の配置

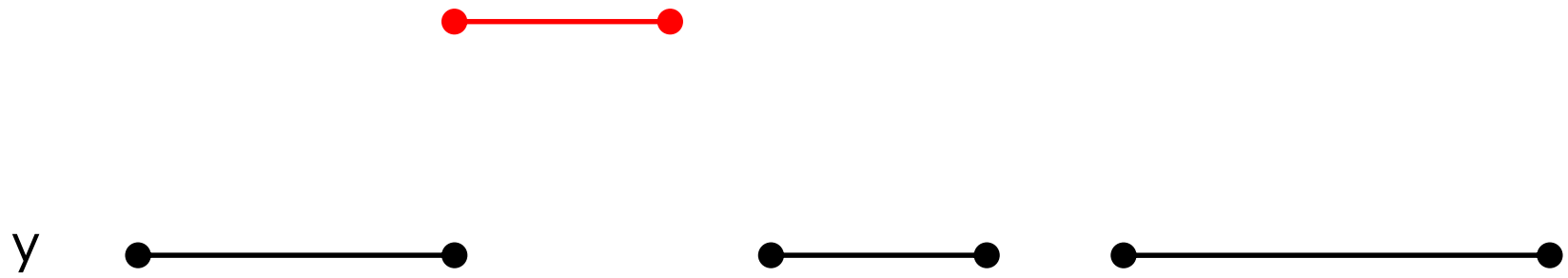
- ２つの問題に分けて考える：与えられた線分について
  - それと平行な線分と触れないか判定.
  - それと垂直な線分と触れないか判定.
  - 両方満たしたら追加する.

問９　線分配置

AIZU
THE UNIVERSITY OF AIZU

平衡二分木

- 平行な線分の判定.

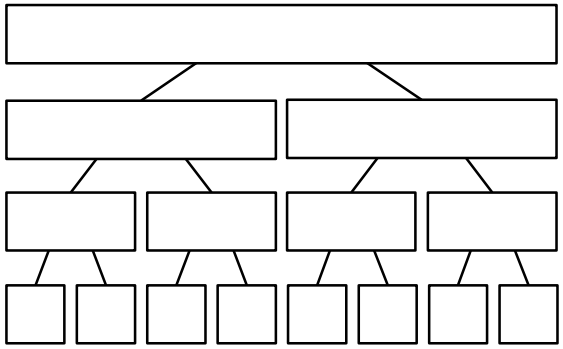


- 各座標ごとに平衡二分木（set）を持ち、配置された線分を保持.
- 二分探索(lower_bound)で与えられた線分が追加できるか判定する.

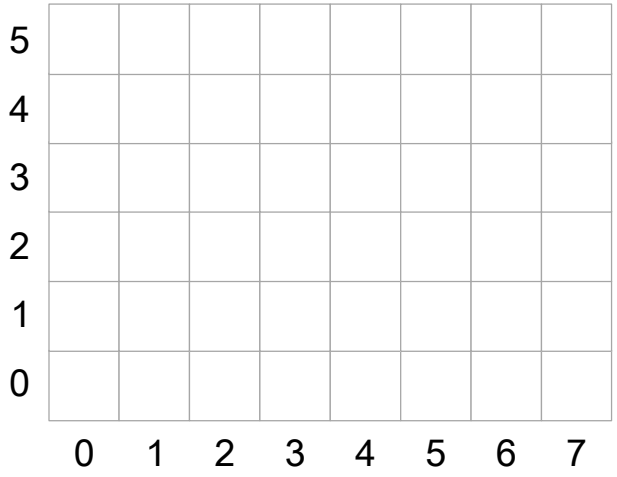
問９　線分配置

セグメントツリー

- 垂直な線分の判定.



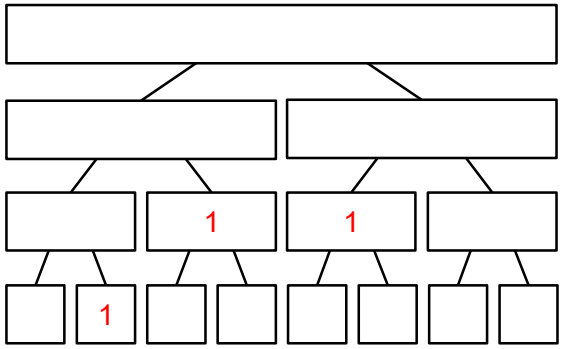
各ノードには、整数の集合を管理するデータ構造（平衡二分木）



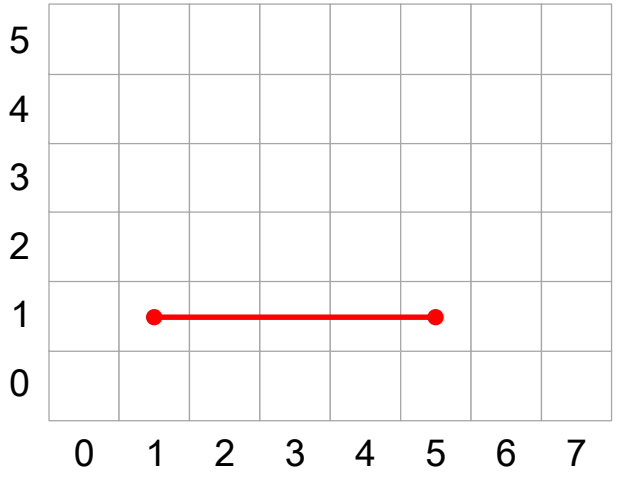
問９　線分配置

セグメントツリー

- 垂直な線分の判定.



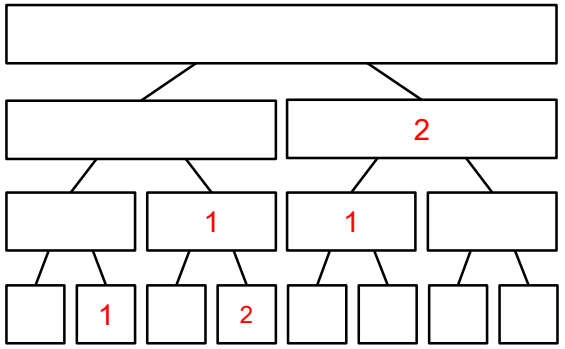
各ノードには、整数の集合を管理するデータ構造（平衡二分木）

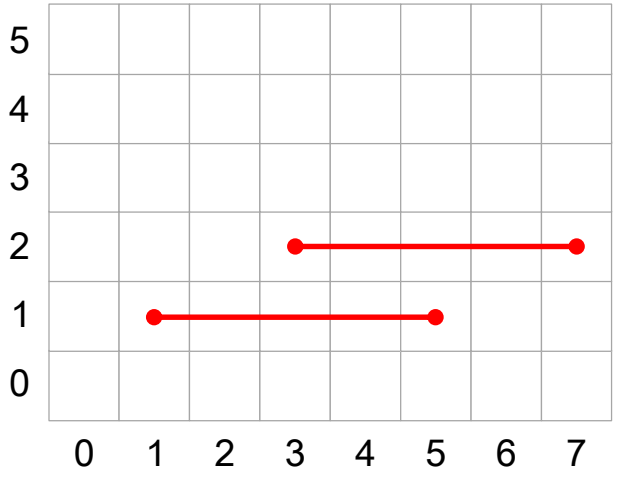


問９　線分配置

セグメントツリー

- 垂直な線分の判定.

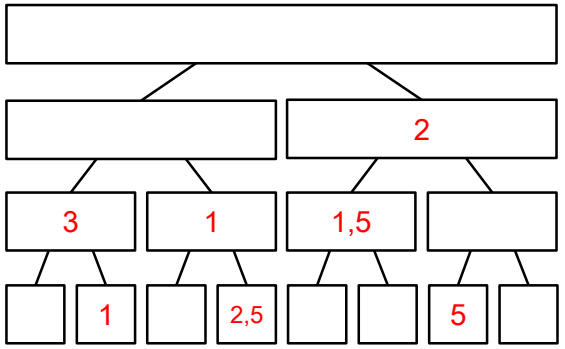


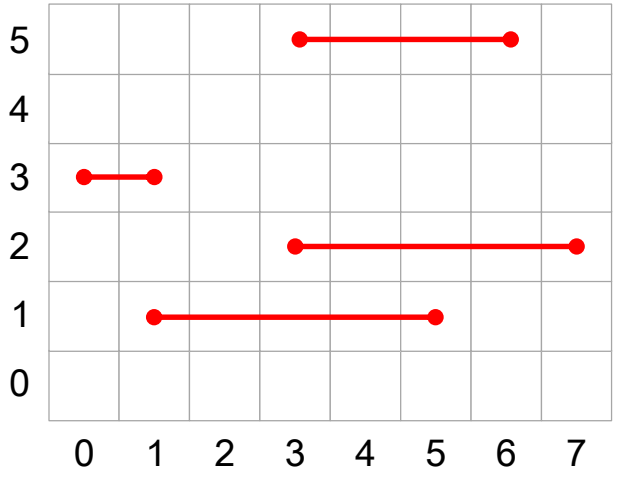


問９　線分配置

セグメントツリー

- 垂直な線分の判定.

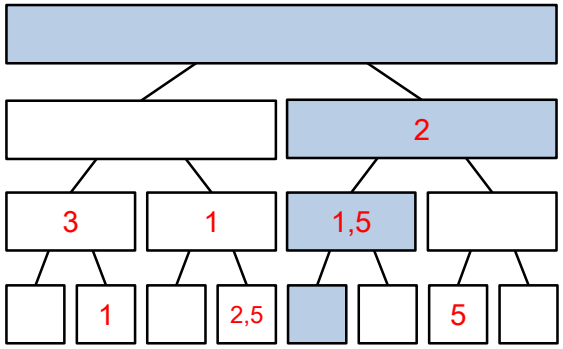




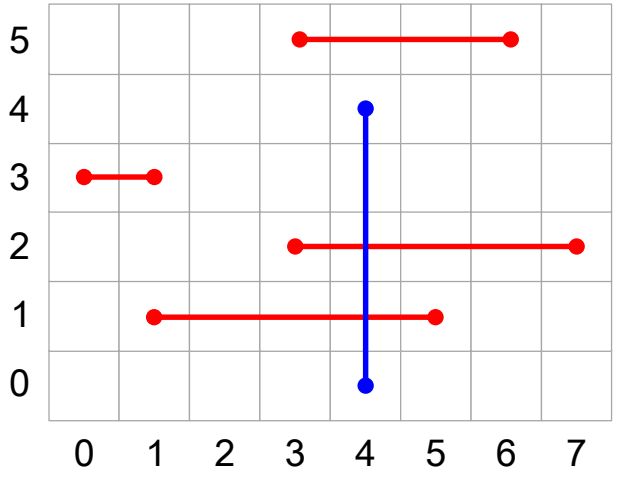
問９　線分配置

セグメントツリー

- 垂直な線分の判定.



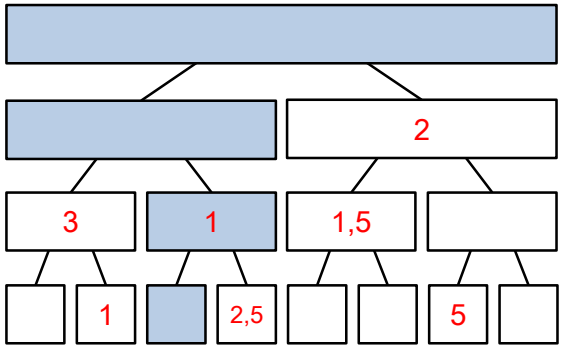
これらのノードにある整数の集合に、0以上4以下の整数があるか判定する.



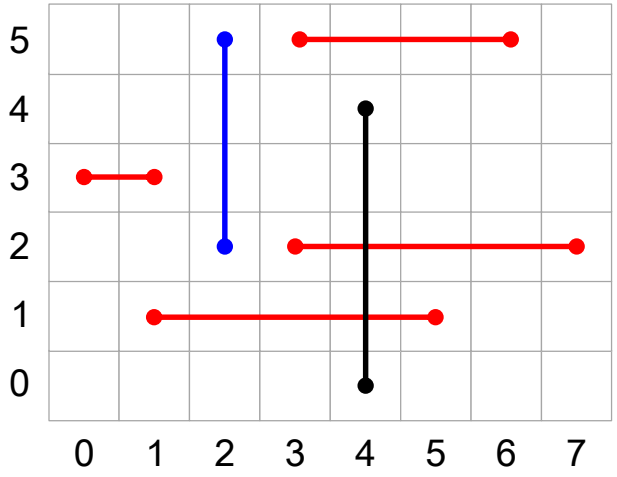
問９　線分配置

セグメントツリー

- 垂直な線分の判定



これらのノードにある整数の集合に、2以上5以下の整数があるか判定する.



問９　線分配置

AIZU
THE UNIVERSITY OF AIZU

セグメントツリー

- 集合の中に、a 以上 b 以下の要素があるかを判定する.
- 二分探索
  - lower_bound(a) と upper_bound(b) が一致するか判定する.

問９　線分配置

AIZU
THE UNIVERSITY OF AIZU

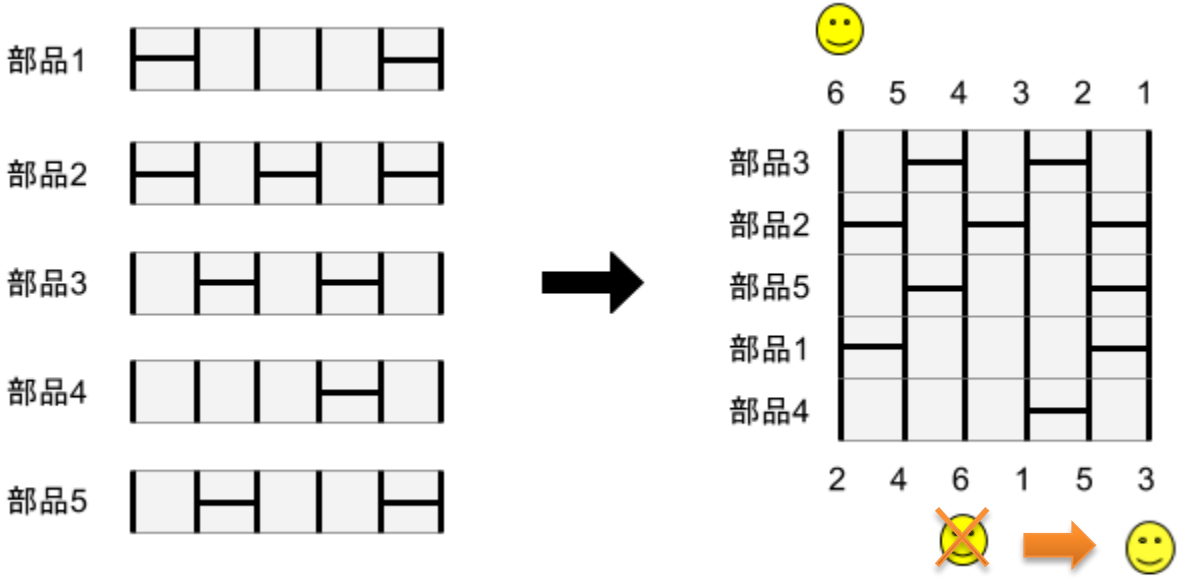
計算量

- セグメントツリーへの要素の追加・探索　O(logN).
- 平衡二分木への要素の追加・探索　O(logN).
- 各線分について、セグメントツリーの中身の平衡二分木へのクエリを行って O(N × logN × logN).

問１０　あみだくじ

問題概要

- あみだくじの部品が与えられる.
- 部品の順序を自由に決定し，左端開始で右端に到達できるか判定せよ.



問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

問題概要

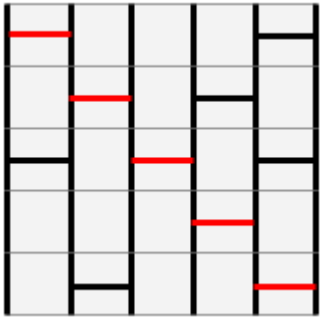
- あみだくじの部品が与えられる.
- 部品の順序を自由に決定し，左端開始で右端に到達できるか判定せよ.
- 到達できる場合はそのような部品の順序のうち，辞書順最小なものを求めよ.
- 制約:
  - 2≤ N(縦棒の本数-1, および，あみだくじの部品の数) ≤ 500.
  - 0≤ M(全部品の横棒の総数) ≤ 10,000.

問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

考察1：右端への到達性

- 常に右に移動するように，部品を並べる必要がある．
  - 左端から右端への移動は，少なくとも，N-1回右に移動する．
  - 移動の機会は段数(N-1)回しかない．
  - 図のような解のパターンしかない．

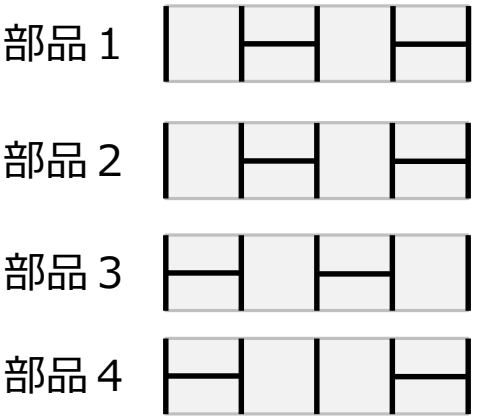


問１０　あみだくじ

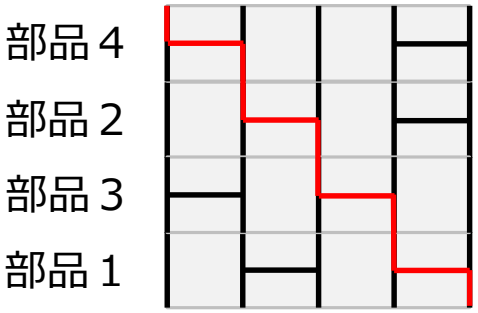
AIZU
THE UNIVERSITY OF AIZU

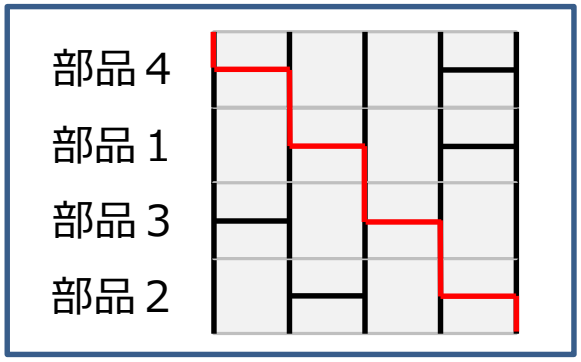
問題概要

入力例２	出力例２
5 0 1 0 1 0 1 0 1 1 0 1 0 1 0 0 1	yes 4 1 3 2



この例では、２通り可能だが、４１３２を選ぶ





問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

考察

- 部品の数がN＝500なので、DFSで可能な組み合わせを全て試すと500！.
- DFS１段階目の探索で１列目に横棒があるパーツだけ探索、２段階目の探索で２列目に横棒があるパーツだけ探索、…としても、まだ全然間に合わない.

問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

考察２：到達性判定

- 先ほどの考察を踏まえると，各部品の配置可否情報が得られる

  i 段目に部品 j を配置可能

  ⇔

  部品 j には i 番目と i+1 番目の縦棒を結ぶ横棒が存在

- 配置可能な情報から，各部品を各場所に割り当てる問題

  ⇒ 二部マッチング

問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

解法: 二部マッチングの辞書順最小

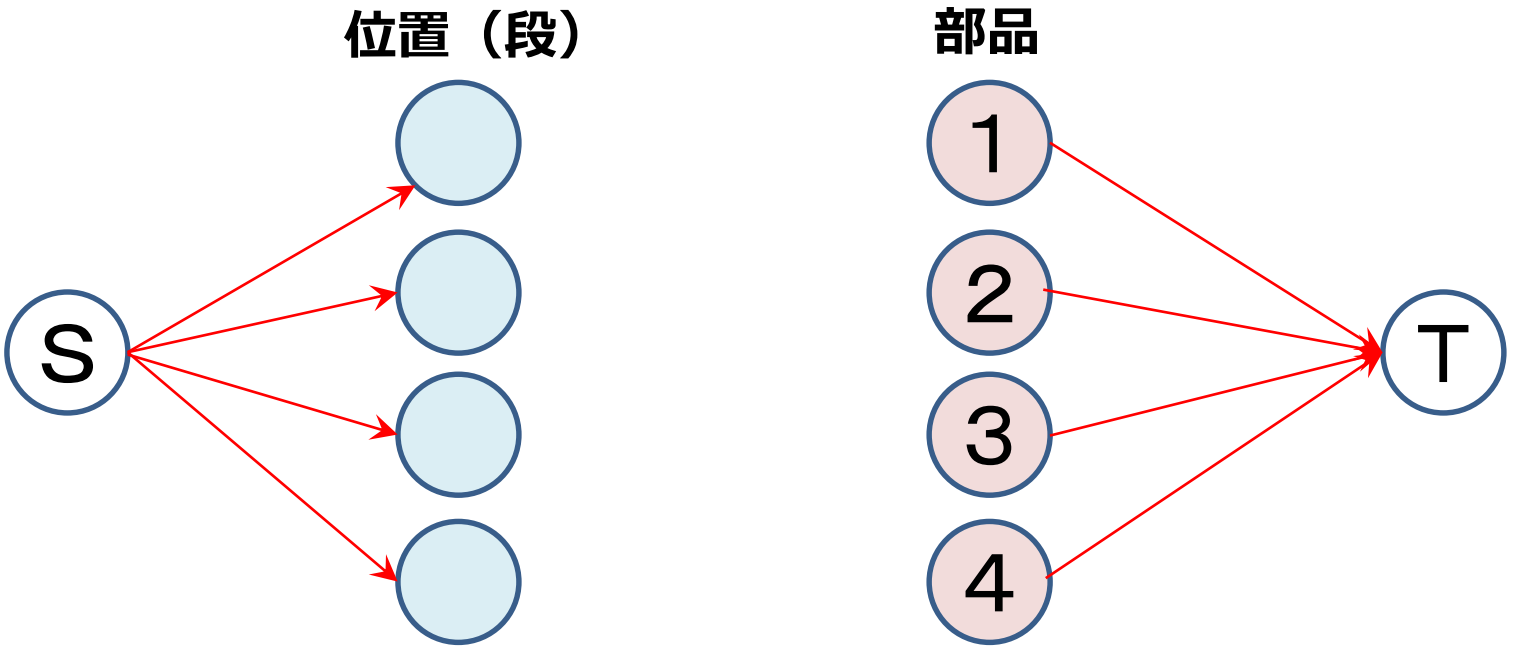
- フローネットワーク上で求めた最大流の解を変形することで求めることが可能.
- 流量を維持したまま辺が削除可能か調べて，ネットワークを変形する.
- 各辺について，流量を変更せずに削除可能か調べる.
- 最も使用されると不利になる辺から貪欲的に削除していく.

問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

解法

- ２部マッチングで、候補をとなる並び順を一つだけ探す（最大流）.
- マッチングのグラフの作り方：
  - 左側（青いノード）と右側(赤いノード）をＮ－１個用意.
  - ソース(ノードS)から青いノード全てに、容量１のエッジを繋ぐ.
  - 赤いノード全てからシンク(ノードＴ）に、容量１のエッジを繋ぐ.

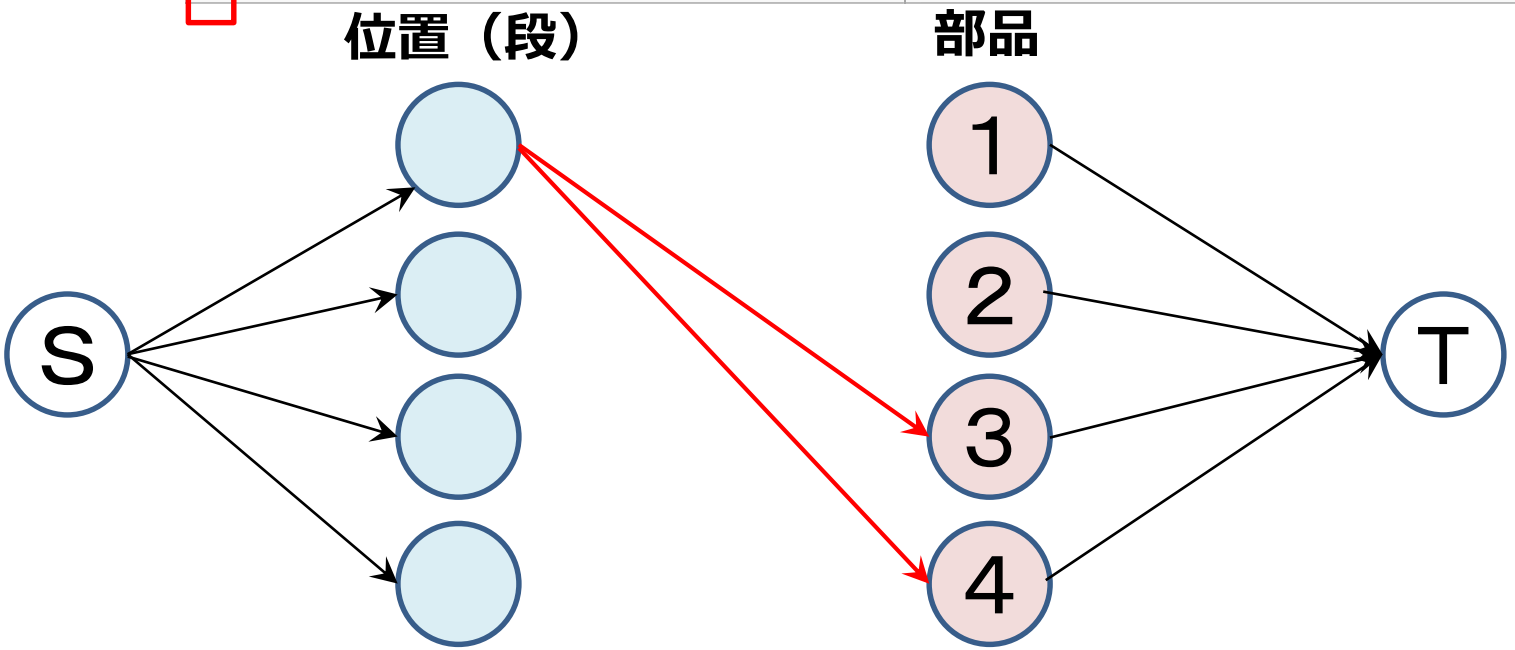


問１０　あみだくじ

解法

- ｉとｉ＋１を繋ぐ横棒がｊ番目の部品にある場合、左のｉ番目と右のｊ番目を容量１のエッジで繋ぐ.
- 例えば

入力例２	出力例２
5 0 1 0 1 0 1 0 1 1 0 1 0 1 0 0 1	yes 4 1 3 2



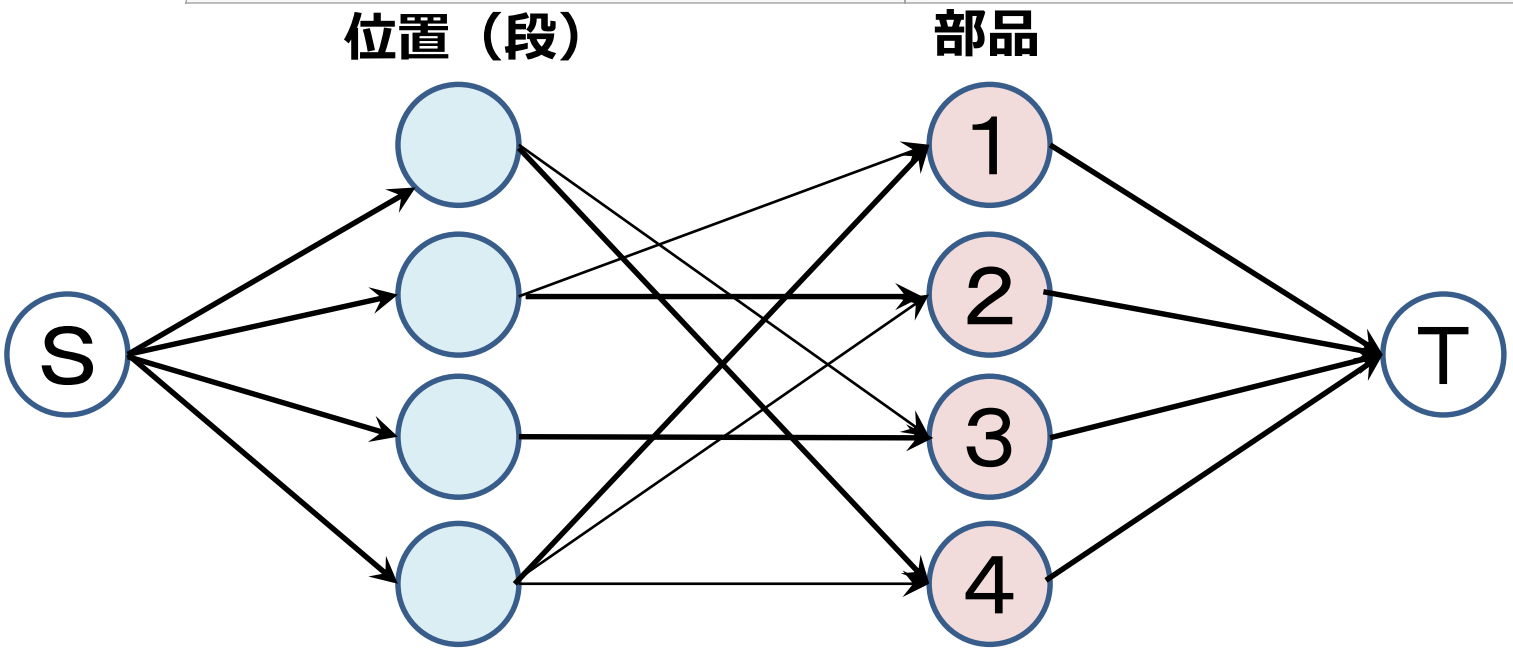
問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

解法

- 最大流が４なら、yes（４　２　３　１のマッチング）.

入力例２	出力例２
5 0 1 0 1 0 1 0 1 1 0 1 0 1 0 0 1	yes 4 1 3 2

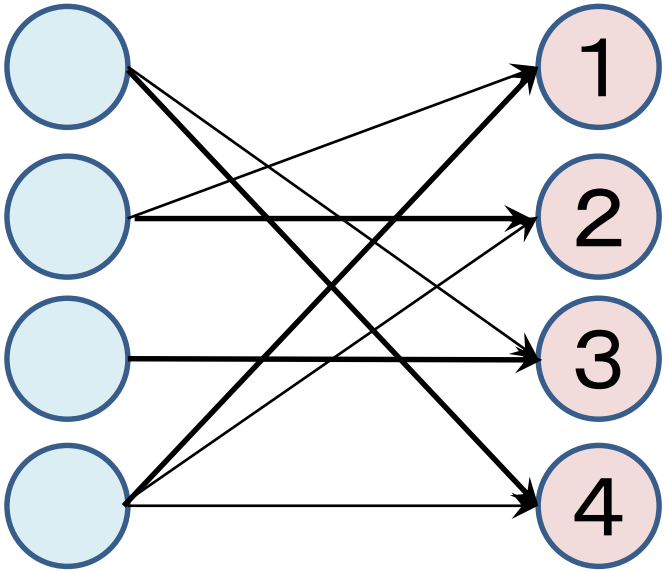


問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

解法

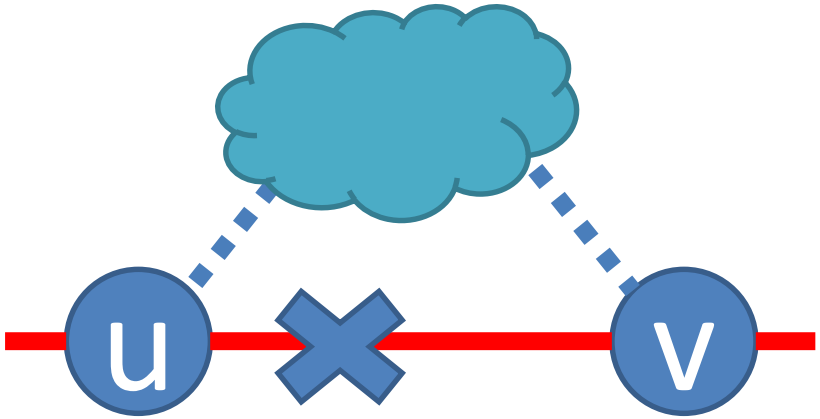
- 辞書順最小のものを探すために、作れる最大流を全て初めから試すと時間制限.
- 最大流を流した後、必要な部分だけ繋ぎ替える.
- 辞書順最小にするには？
  - １番上に置く部品に、より部品番号が小さいものが使えないか？
  - ２番目に上に置く部品に、より部品番号が小さいものが使えないか？
  - …というのをくりかえしていく.



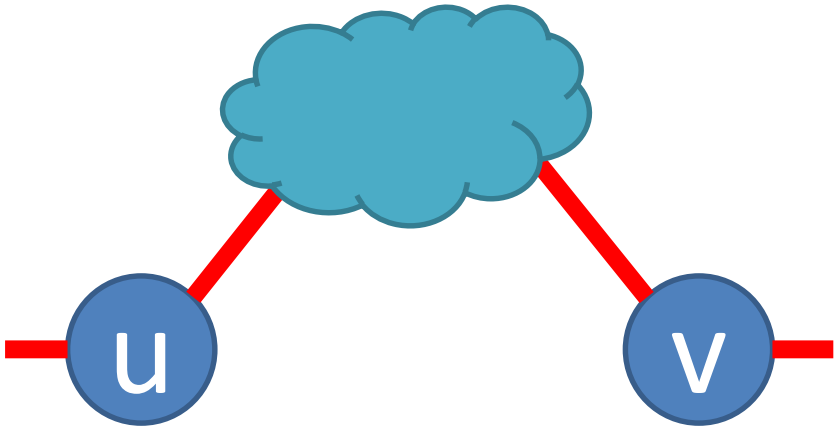
問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

解法: 辺の削除



削除したい辺, u, v



残余グラフで(u, v)に追加で
フローを流して，その後削除

流せない場合，流量が変
わってしまうため削除不可

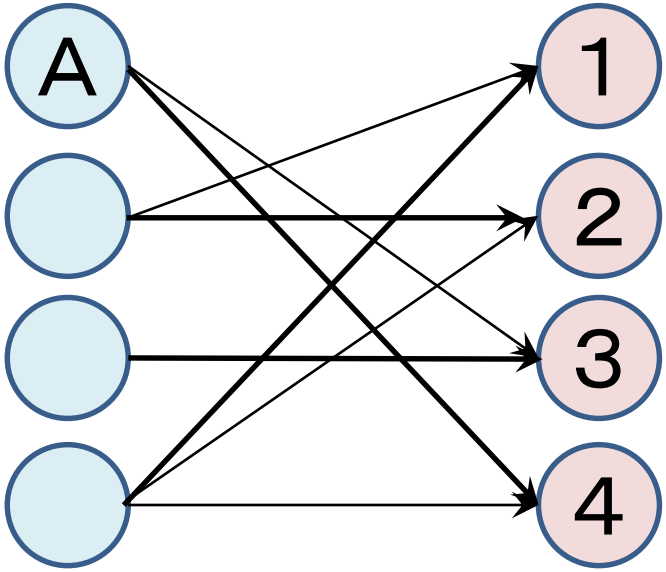
削除するとフロー保存則を満
たした状態になる

問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

解法

- １番上に置く部品に、より部品番号が小さいものが使えないか？
  - 左の節点の一番上から、右の節点の一番下から上に順番に、繋ぐエッジを見ていく.
  - フローが流れていなければ、そのエッジを今後使うことはないので、容量を０にしておく.
  - フローが流れている場合、その２点間でフローを流す経路が無いか調べる.
  - 別の経路で流せれば、より下側に部品番号が大きいものが置けるということ.

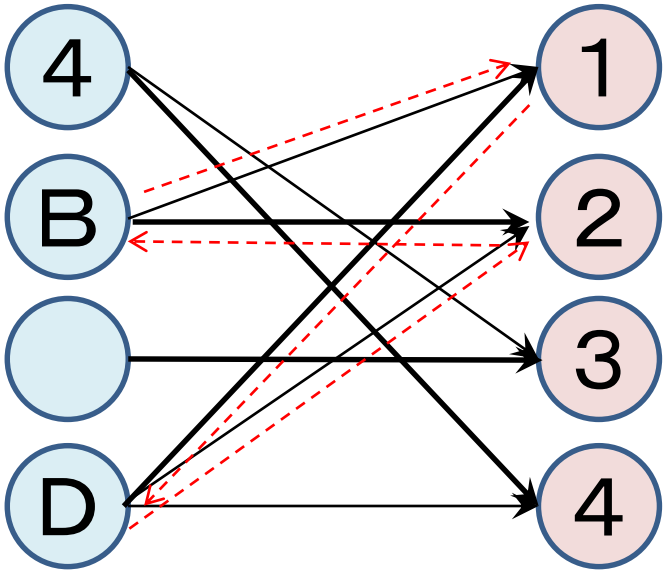


- Ａ → ４は流れているので他のつなぎ方が無いか調べる.
- フローが流れていない辺と、逆辺（フローが流れている辺を逆向きにしたもの）を使った、Ａ → ４への経路が他に無い.
- つまり、４番目の部品より小さな番号の部品を１段目に置けないので、１段目は４番目の部品にするしかない.

問１０　あみだくじ

解法

- ２番目に上に置く部品に、より部品番号が小さいものが使えないか？

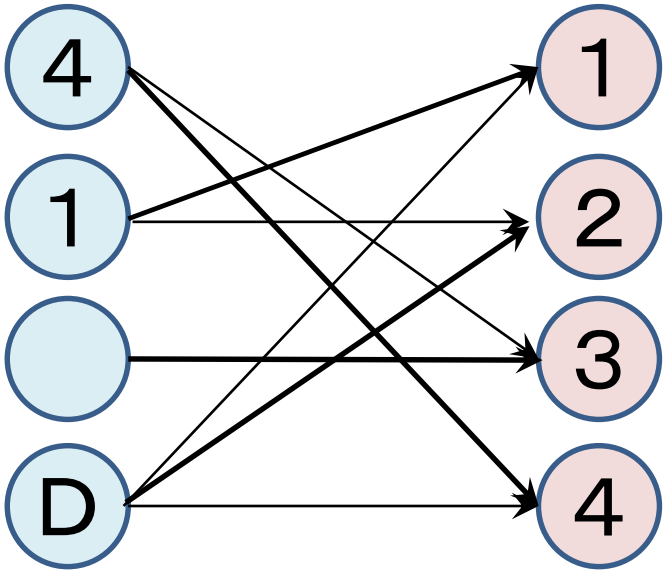


- Ｂ → ２は流れているので他のつなぎ方が無いか調べる.
- フローが流れていない辺と、逆辺（フローが流れている辺を逆向きにしたもの）を使った、Ｂ → ２への経路にバイパス、Ｂ→１→Ｄ→２が存在する.
- つまり、２番目の部品より小さな番号の部品１番を２段目に置けるので、フローグラフをＢ→２はＢ→１、Ｄ→１はＤ→２と繋ぎ替える.

問１０　あみだくじ

解法

- ２番目に上に置く部品に、より部品番号が小さいものが使えないか？



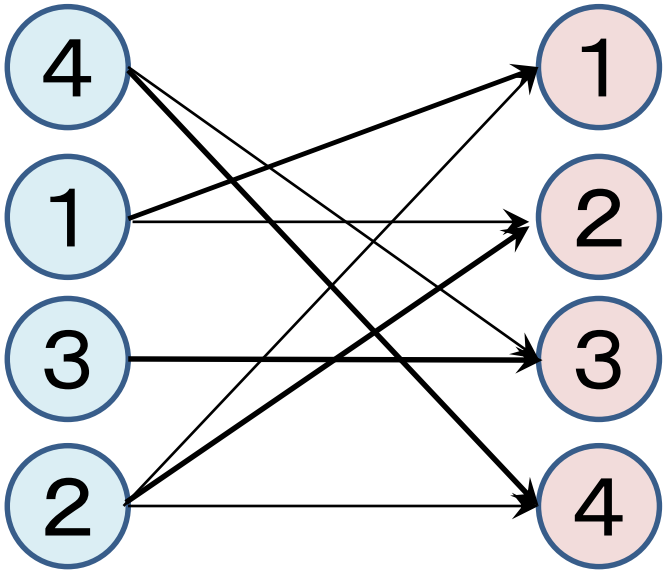
- Ｂ → ２は流れているので他のつなぎ方が無いか調べる.
- フローが流れていない辺と、逆辺（フローが流れている辺を逆向きにしたもの）を使った、Ｂ → ２への経路にバイパス、Ｂ→１→Ｄ→２が存在する.
- つまり、２番目の部品より小さな番号の部品１番を２段目に置けるので、フローグラフをＢ→２はＢ→１、Ｄ→１はＤ→２と繋ぎ替える.
- ２段目確定.

問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

解法

- この処理を一番下の段まで行う.
- 「４　２　３　１」が「４　１　３　２」に改善.



問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

解答例（疑似コード）

```
bool i段目に部品jを置く必要があるか?{
  段iと部品jの辺を削除しても、流量が変わらないか?
}

void 辞書順最小を求める{
  for i: 1 → 段数{
    for j: 部品 → 1{
      if( i段目に部品jを置く必要があるか?){
        jはiに置くことが決定
      }}}


bool 段iと部品jの辺を削除しても、流量が変わらないか?{
  ret = 段iの頂点から部品jの頂点へ残余グラフで流せるか?
  if(ret) 流せるなら流し、該当辺を削除
  return ret
}
```

問１０　あみだくじ

AIZU
THE UNIVERSITY OF AIZU

計算量

- 参加人数N、横棒の総数E.
- 前計算(二部マッチング).
  - $O(N \times E)$.
- 辞書順最小を求める.
  - $O(E^2)$.