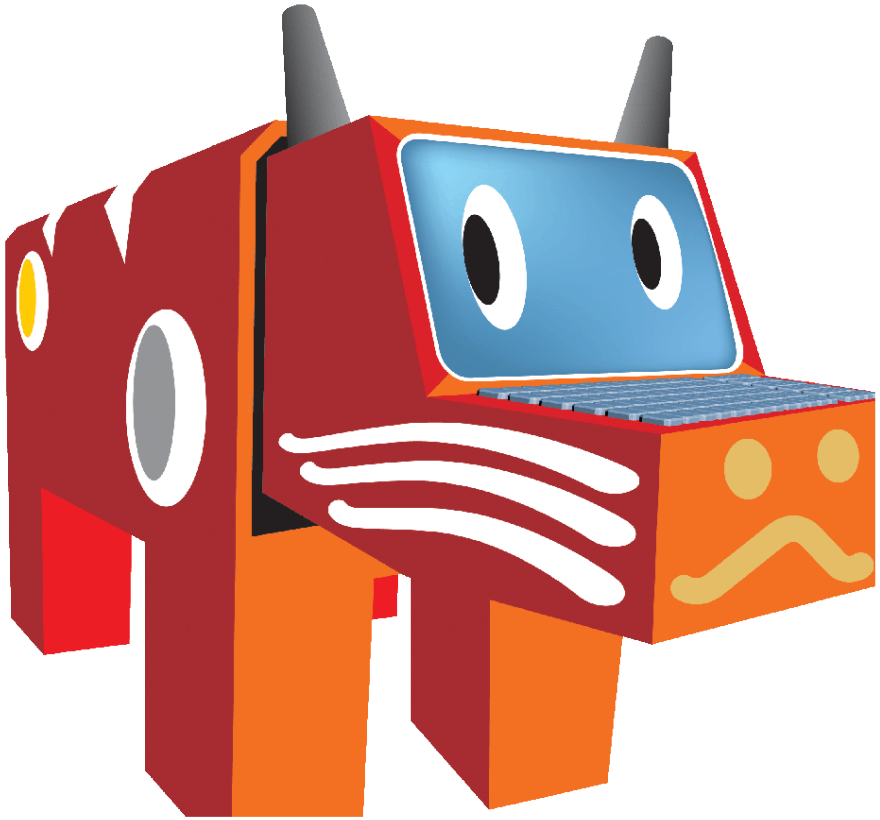
パソコン甲子園2013本選

プログラミング部門　解説

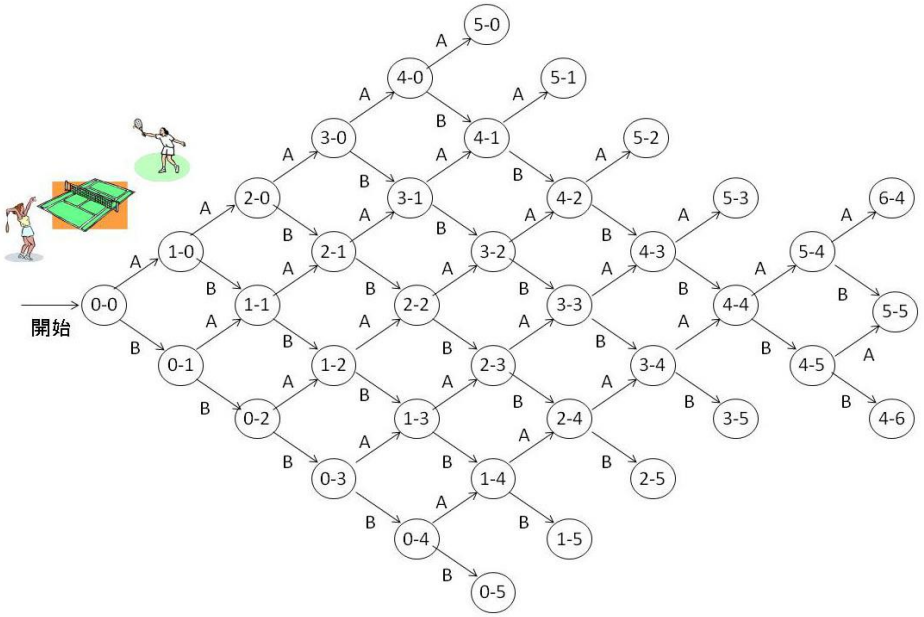


会津大学

問1　テニス

問題概要

- テニスの得点が図のように遷移する.
- 得点を0-0から始めて、指定された得点に至る経路をすべて列挙せよ.



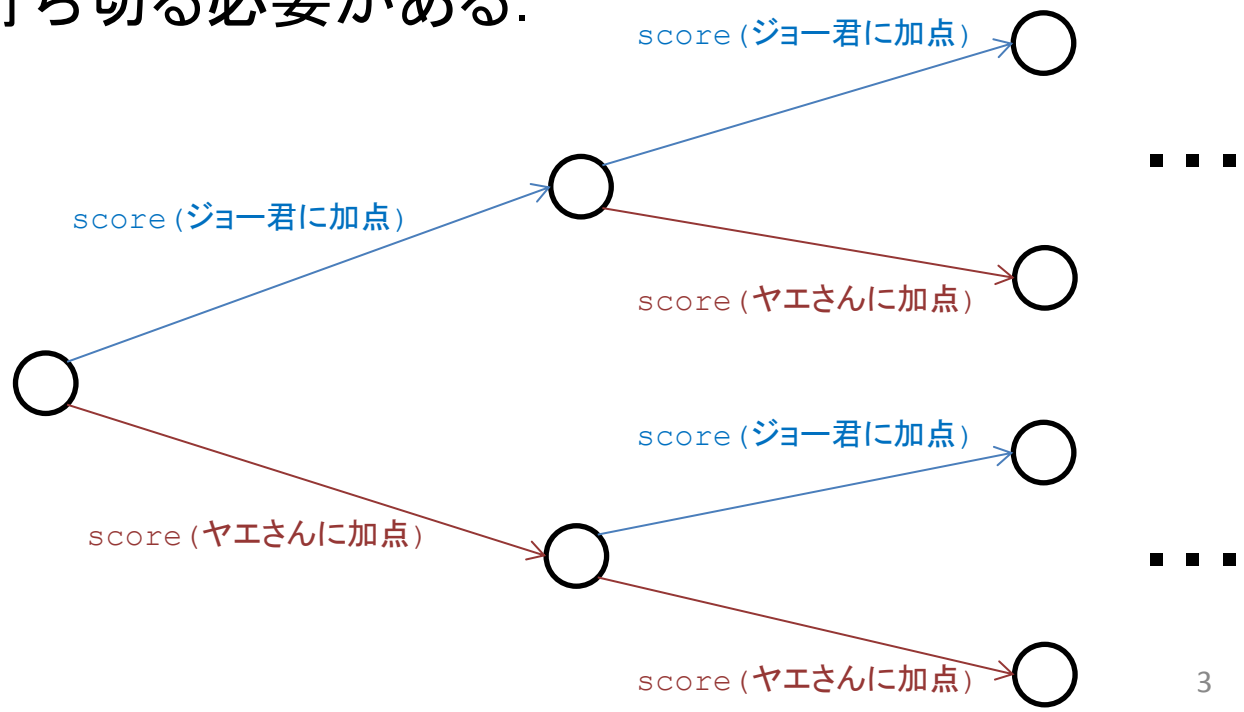
講評

- 組み合わせ・パタンを列挙するプログラミング技法が必要.

問1　テニス

解法

- 再帰によって全ての経路を列挙する.
  - ある関数で再びその関数を呼び出すプログラミング技法
- ジョー君、ヤエさんそれぞれへの加点で分岐.
- 適切に再帰を打ち切る必要がある.



問1　テニス

C++による解答例

```
void score(int j, int y, string p){
    if ( j == J && y == Y ){
        cout << p << endl; return;
    }
    if ( j == 5 && y <= 3 || y == 5 && j <= 3) return;
    if ( j > J || y > Y ) return;
    score(j+1, y, p + "A");
    score(j, y+1, p + "B");
}

main(){
    cin >> J >> Y;
    score(0, 0, "");
}
```

問2　親方の給料計算

問題概要

- 職人の種類と工事の種類が与えられる.
- 工事の単価がいくつか与えられる.
- 次の計算（親方が書いたプログラム）で職人の給料を求める.

```
int i, j;
for ( i = 0; i < N; i++ ){
    c[i] = 0;
    for ( j = 0; j < M; j++ ){
        c[i] = c[i] + a[i][j]*b[j];
    }
}
```

- 職人の人数N≦10000、工事の種類の数M≦10000.
- ただし、職人が工事をこなした総数は50000以下.

問2　親方の給料計算

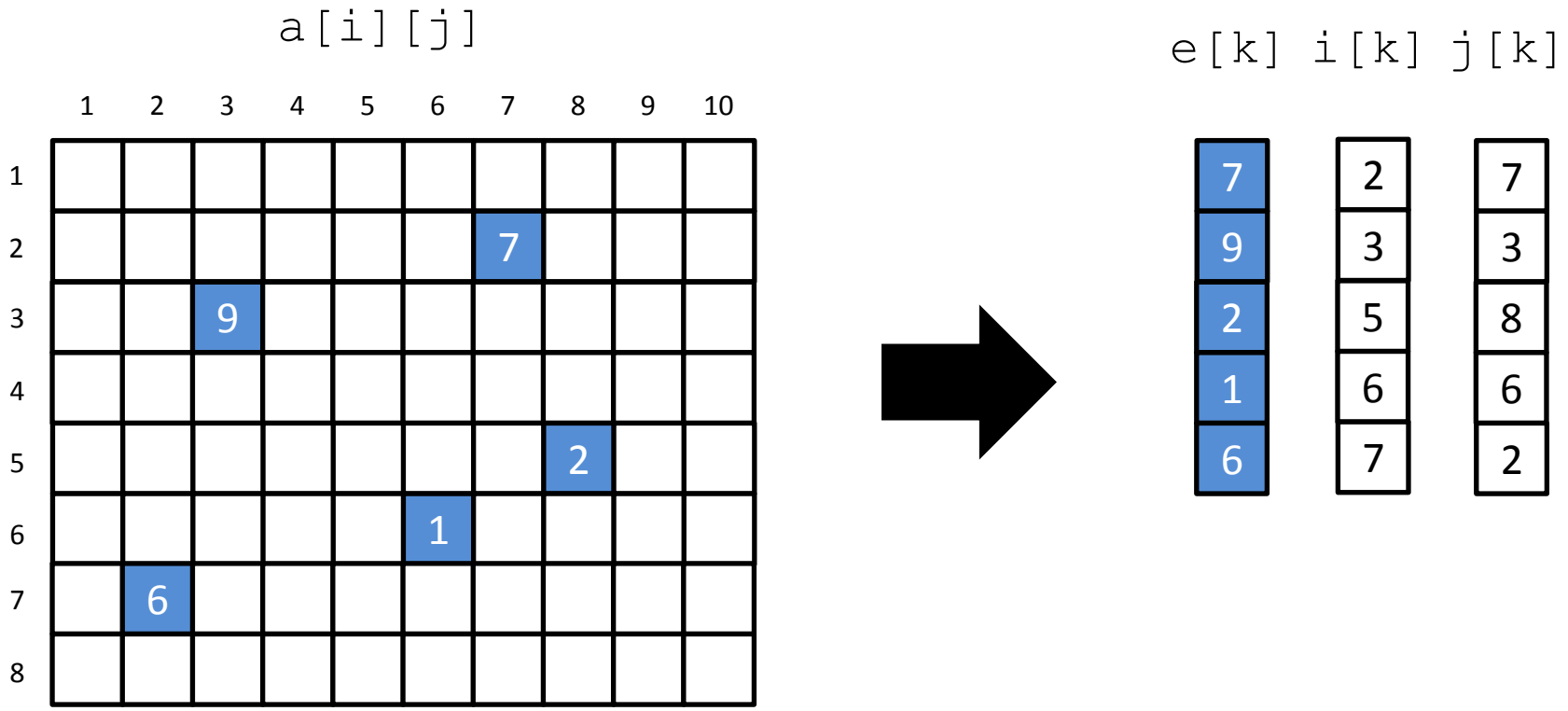
講評

- 入力値の最大ケースを考慮すると、親方が書いたプログラムでは効率が悪い.
- 無駄を無くしプログラムを高速化する必要がある.
- 適切なデータ構造とアルゴリズムに変更できるかが問われている.

問2　親方の給料計算

解法

- k番目に現れたデータを e[k], i[k], j[k]として1次元配列に格納.
- （e[k]は2次元配列a[i][j]の代わり）



問2　親方の給料計算

解法

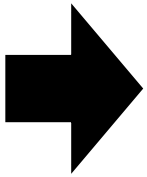
- 親方のプログラム中の a[i][j] は、表のすべての要素を持つ.
  - 最悪10000×10000回の計算

```
for ( i = 0; i < N; i++ ){
    c[i] = 0;
    for ( j = 0; j < M; j++ ){
        c[i] = c[i] + a[i][j]*b[j];
    }
}
```

N * M = 100,000,000

- 値のある要素だけを持つように変更する.
  - 題意より最大5万回の計算

eiの合計が5万
ei≧1で整数
つまり1≦i≦50000



```
for(k = 0; k < K; k++ ){
    c[i[k]] += e[k] * b[j[k]]
}
```

K = 50,000

問3　塵劫記

問題概要

- m(2≦m≦20)を基数、n(1≦n＜240)を指数とした m^n を日本の数の単位で表す.
- 最大10^{72}(無量大数).

講評

- int(最大21億程度),long long(最大922京)等では表現できない.
- doubleは10^{308}程度だが、有効桁数が15～16程度しかない.
- 多倍長整数の乗算を実装できるかが問われている.

問3　塵劫記

解法

- 単位は17種類＋10000未満.
  → 要素が18個のint配列でブロック分けする.

無量大数	不可思議	那由他	阿僧祇	恒河沙	極	載	正	澗	溝	穣	秭	垓	京	兆	億	万
9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0	9999～0

1. 10000未満の数値を基数mで初期化.
2. 全ブロックにmを掛ける.
3. 各ブロックで10000以上になったものは一つ上のブロックに繰り上げる.
4. 手順2,3をn-1回繰り返す.

問3　塵劫記

解法

- 例：m = 5 のとき n = 11 から n = 12 へ

億	万	1万未満
0	4882	8125

億	万	1万未満
0	24410	40625

億	万	1万未満
0	24414	625

億	万	1万未満
2	4414	625

問3　塵劫記

Javaによる解答例

```
Stirng[] table = {"", "Man", "Oku", "Cho", "Kei", "Gai", "Jo",
                        "Jou", "Ko", "Kan", "Sei", "Sai", "Gok",
                        "Ggs", "Asg", "Nyt", "Fks", "Mts"};//18種類

ArrayList<Integer> aBlock = new ArrayList<Integer>();
BigInteger val = BigInteger.valueOf(m).pow(n); //m^n
BigInteger tts = BigInteger.valueOf(10000);//ブロック区切り

while (val != BigInteger.ZERO) {
    aBlock.add(val.mod(tts).intValue());
    val = val.divide(tts);
}

for (int i=aBlock.size()-1; i>=0; --i) {
    if (aBlock.get(i) != 0)
        System.out.print(aBlock.get(i) + table[i]);
}
System.out.print("¥n");
```

問3　塵劫記

C++による解答例

```
const char* table[] = {"", "Man", "Oku", "Cho", "Kei", "Gai", "Jo",
                       "Jou", "Ko", "Kan", "Sei", "Sai", "Gok", "Ggs",
                       "Asg", "Nyt", "Fks", "Mts"};//18種類

int block[18]; //←すべて0に初期化
block[0] = m;
for (int i=1; i<n; ++i) {
  for (int j=0; j<18; ++j) block[j] *= m; //全ブロックにmを掛ける
  for (int j=0; j<18; ++j) {//繰り上がり処理
     if (block[j] >= 10000) {
        int nCarry = block[j]/10000;//繰り上がり分
        block[j+1] += nCarry;
        block[j] -= nCarry*10000;
     }
  }
}
//出力
for (int i=17; i>=0; --i)
  if (block[i]) printf("%d%s", block[i], table[i]);
printf("¥n");
```

問4　巨樹の刻み手

問題概要

- 耐久力 D の巨樹を N 種類の道具を使ってたたく.
- 道具 i でたたくと、巨樹の耐久力が ai 減り ei の経験値を得る.
- ただし、道具 i を使用するには ri の経験値が必要.
- D を 0 にするたたく回数の最小値を求めよ.

解法１

- （巨樹の耐久力、現在の経験値）を頂点とした幅優先探索.

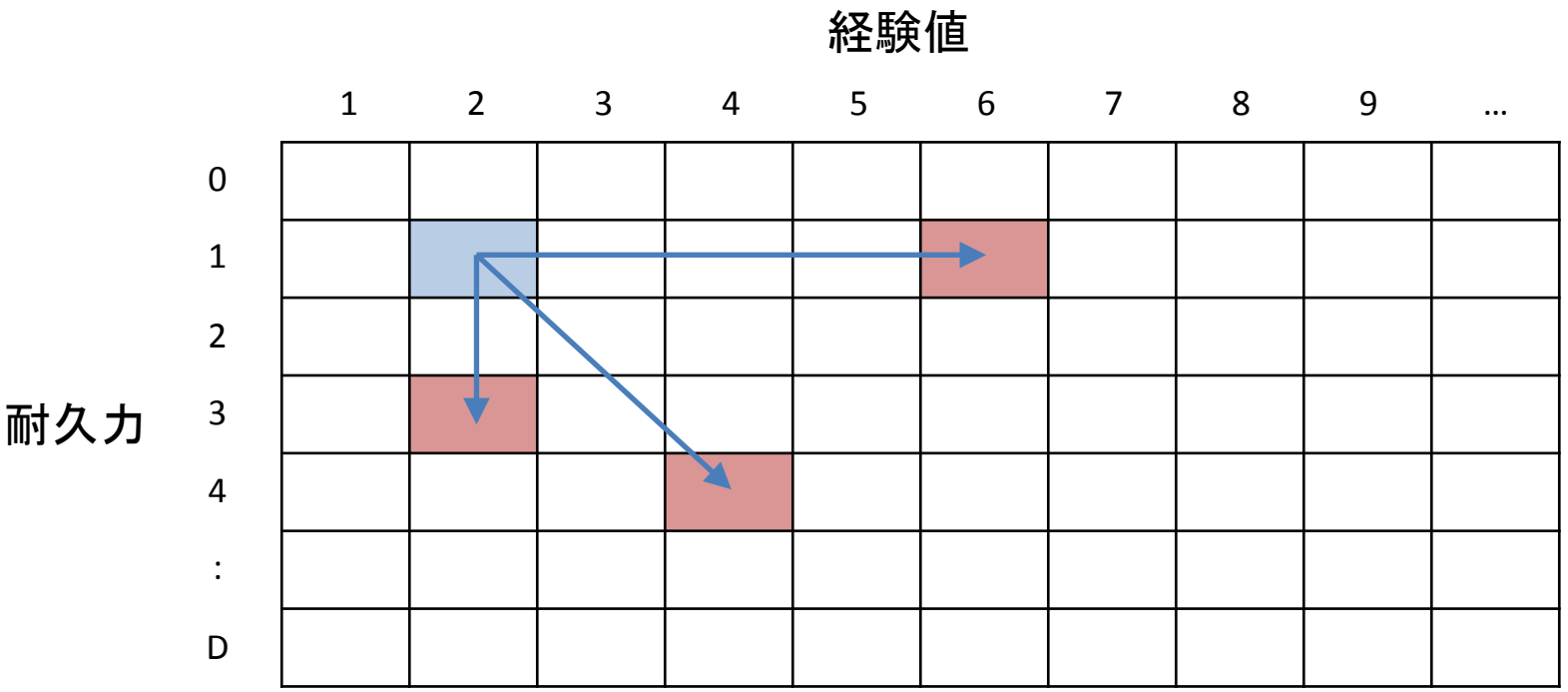
解法２

- （巨樹の耐久力、現在の経験値）を状態とした動的計画法.

問4　巨樹の刻み手

解法1，2

- （巨樹の耐久力、現在の経験値）を頂点（状態）とする.
- 現在の経験値で使用できる道具を使って、頂点を移動.




```
dp[ne][nd] = min( dp[ne][nd], dp[e][d] + 1);
```

問4　巨樹の刻み手

C++による解答例（幅優先探索）

```
queue<pair<int, int> > Q;
C[0][0] = 0;
Q.push(make_pair(0, 0));

while(!Q.empty()){
    u = Q.front(); Q.pop();
    if ( u.first >= D ) return C[u.first][u.second];

    for ( int i = 0; i < N; i++ ){
        if ( I[i].r > u.second ) continue;
        int nd = min(u.first + I[i].a, D);
        int ne = min(u.second + I[i].e, MAX);
        if ( C[nd][ne] == INF ){
              C[nd][ne] = C[u.first][u.second] + 1;
              Q.push(make_pair(nd, ne));
        }
    }
}
```

問4　巨樹の刻み手

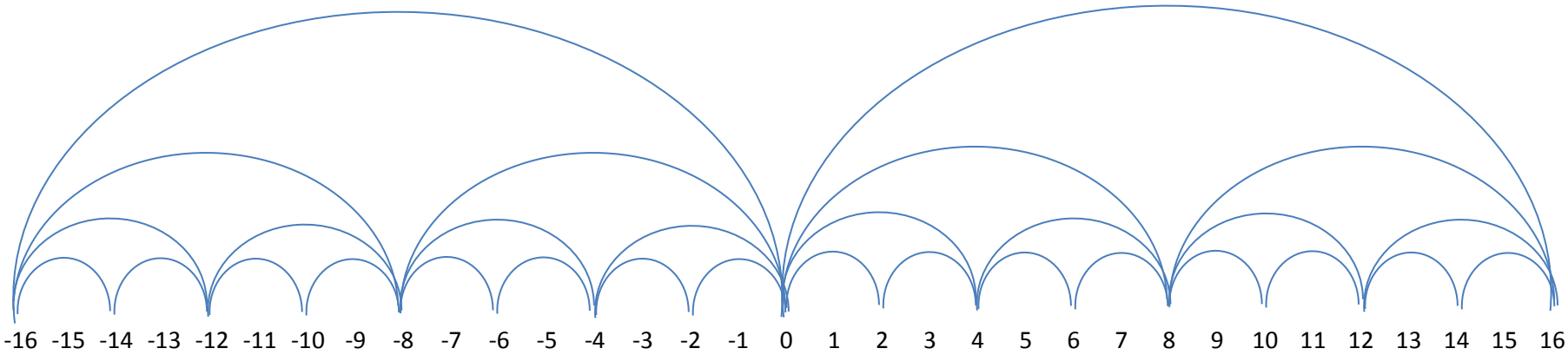
C++による解答例（動的計画法）

```
dp[0][0] = 0;

for ( int d = 0; d <= D; d++ ){
    for ( int e = 0; e <= MAX; e++ ){
        for ( int k = 0; k < N; k++ ){
            if ( dp[e][d] == INFTY ) continue;
            if ( I[k].r > e ) continue;
            int ne = min(e + I[k].e, MAX);
            int nd = min(d + I[k].a, D);
            dp[ne][nd] = min(dp[ne][nd], dp[e][d] + 1);
        }
    }
}
```

問5　無限急行

問題概要

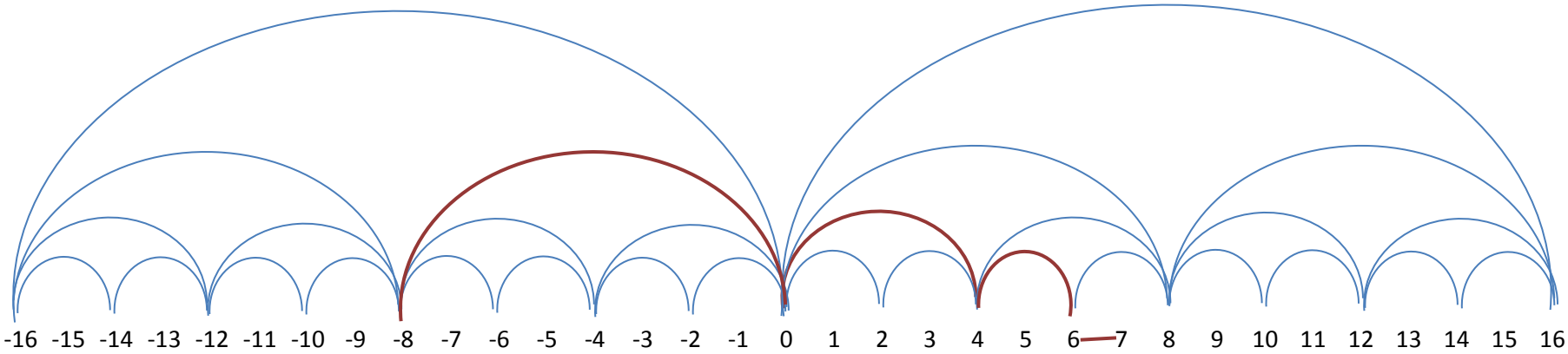
- 2^nの倍数の番号の駅に停車するｎ級快速がある.
- ｓからｄへ移動するための最小の時間を求めよ.



問5　無限急行

問題概要

- 2^nの倍数の番号の駅に停車するｎ級快速がある.
- ｓからｄへ移動するための最小の時間を求めよ.



例：　-8 → 7　　4ステップ

問5　無限急行

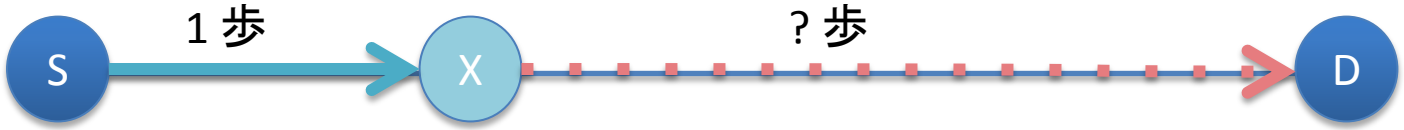
解法1: 貪欲法によるアプローチ

- 「目的地を通り過ぎないような最も高い級の電車に乗って1停車駅分進む」という戦略をひたすら繰り返す.

問5　無限急行

解法1:証明の概略(1)

- 「最初の移動が『目的地を通り過ぎないような最も遠くへの移動』であるような最短経路が少なくとも一つ存在する」ことを示せればよい.

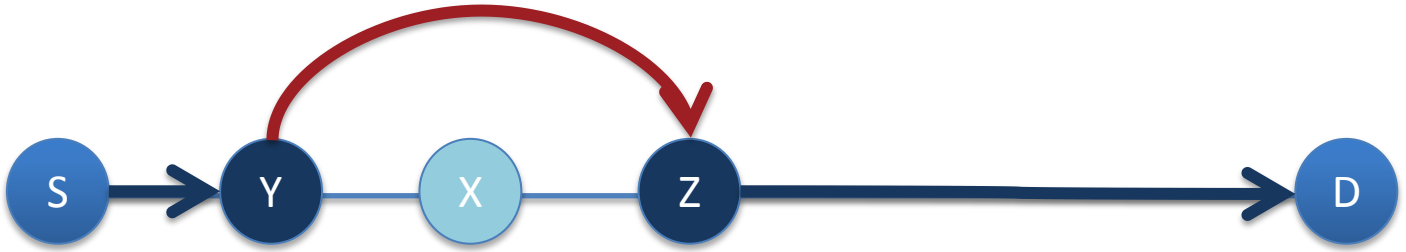


⇒そうでない最短経路を仮定し、矛盾を導く(背理法).

問5　無限急行

解法1:証明の概略(2)

- X で停車しないような経路を考る.



- このような経路はありえない.
- ある駅の “ランク” を、その駅に止まる最も級が高い急行の級数 とする
  - 補題: 駅 a, b, c が順番に並んでいて、Rank(a) < Rank(b) > Rank(c) ならば、駅 a から c までは一度に行けない.
  - 補題: 現在地から駅 X の間に、X よりも高いランクの駅は存在しない.
- 他に考慮すべき経路パターンはいろいろあるが、適切に場合分けして証明する. (ここでは最も複雑なケースについて解説)

問5　無限急行

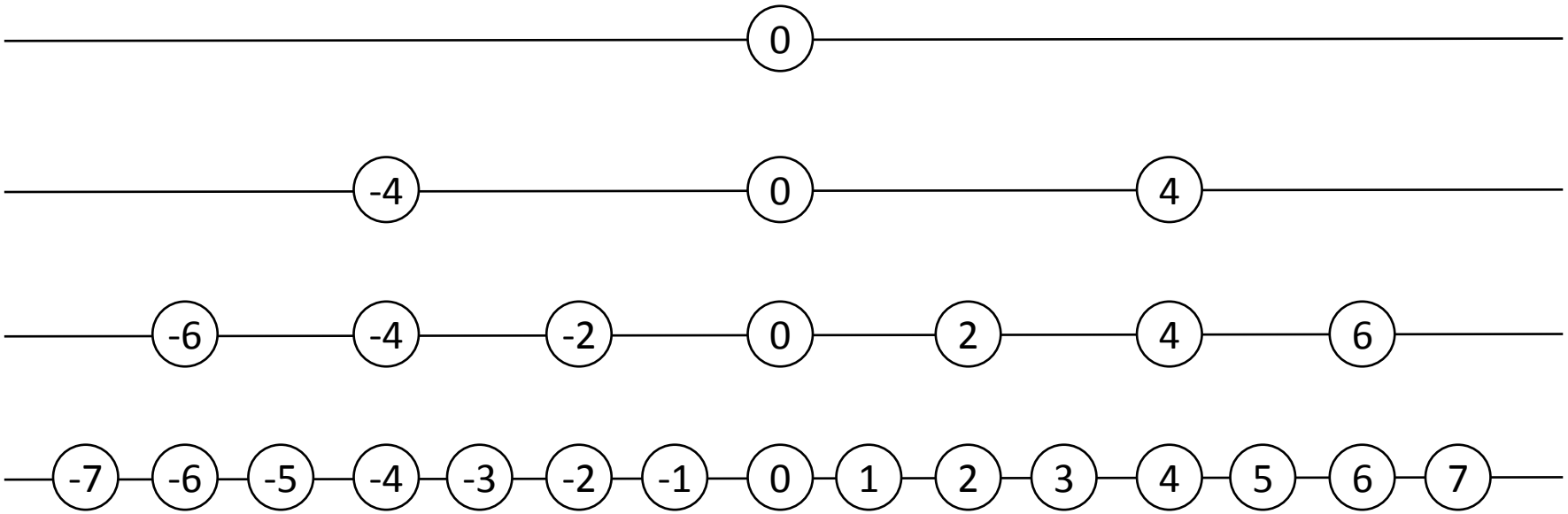
C++による解答例

```
int solve(int s, int d){
    for ( int cnt = 0, cur = s ;;){
        if ( cur == d ) return cnt;
        int p = 1;
        for ( int k = 1; cur + k <= d; k *= 2 ){
            if ( cur % k == 0 ){
                p = k;
            }
        }
        cnt++;
        cur += p;
    }
}
```

問5　無限急行

解法2：　無限の性質をうまく利用する

- 路線図



問5　無限急行

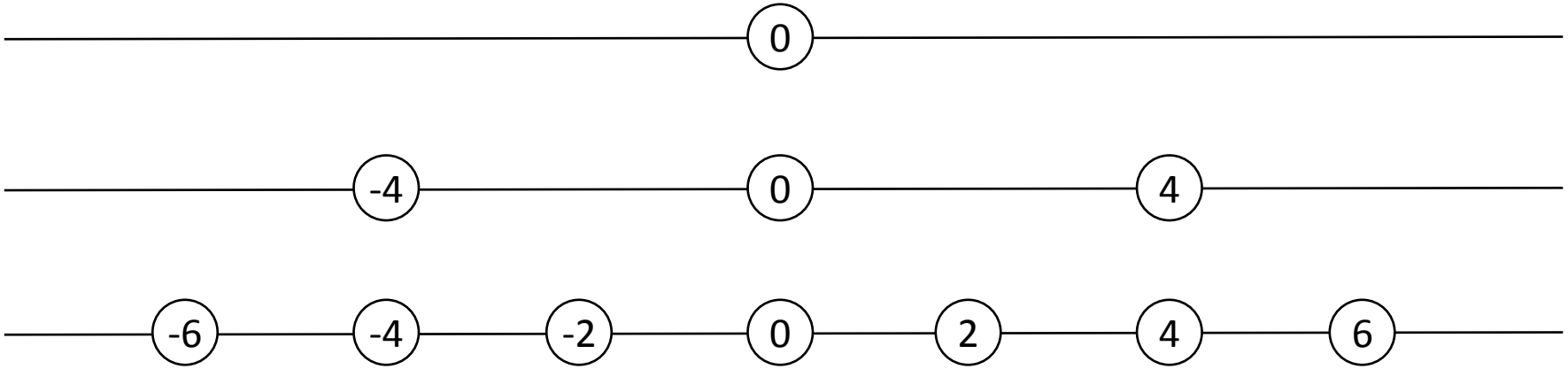
解法2:　無限の性質をうまく利用する

- 補題: 最初の駅の番号が奇数ならば、最初の移動は必ず 0 級快速で隣の駅に進む.
- 補題: 目的の駅の番号が奇数ならば、最後の移動は必ず 0 級快速で直前の駅から進むことである.
- 補題: これらを除けば、0 級快速は一切使用しなくてよい.

問5　無限急行

解法2：　無限の性質をうまく利用する

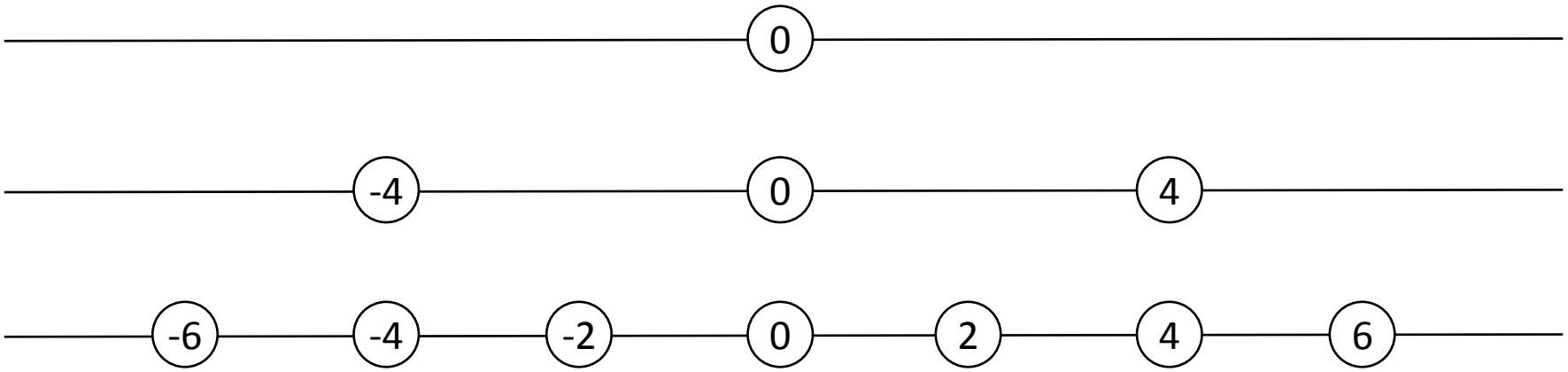
- 0 級快速は必要ない.
- 0 級快速しか停まらない、奇数駅も必要ない.



問5　無限急行

解法2:　無限の性質をうまく利用する

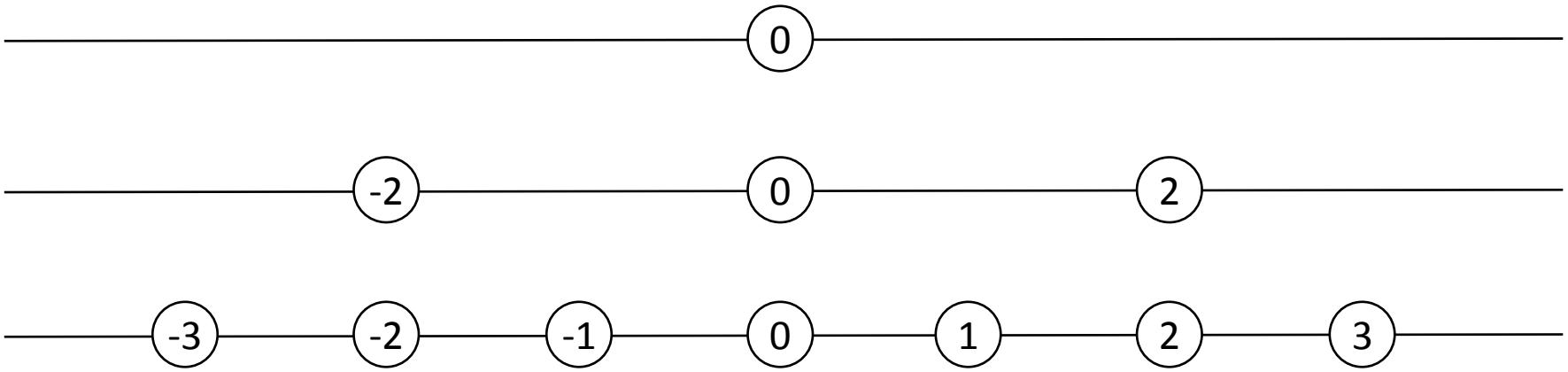
- 1 級快速から 3 級快速までの路線図.



問5　無限急行

解法2:　無限の性質をうまく利用する

- 0 級快速から 2 級快速までの路線図.



- 全ての駅の番号を 2 で割ると見覚えのある問題が現れる.
- 再帰的に解くことが可能.

問5　無限急行

Cによる解答例

```
/**
 * src から dst までの最短距離を求める
 */
int solve(int src, int dst) {
  if (src == dst) return 0;
  int answer = 0;
  if (src % 2 != 0) { answer++; src++; }
  if (dst % 2 != 0) { answer++; dst--; }
  answer += solve(src / 2, dst / 2);
  return answer;
}
```

※　src % 2 == 1 と書いてはいけない！ (-3) % 2 は -1 になる

問6：微生物発電

問題概要

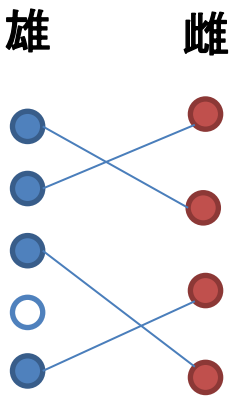
- M 匹の雄の微生物と W 匹の雌の微生物がいる.
- bm[i] と bw[i] が合体すると f(bm[i], bw[i])の電気エネルギーが得られる.
- 微生物の団体のエネルギーの最大値を求めよ.

講評

- 全ての組み合わせを試すと

  12! = 479,001,600通り

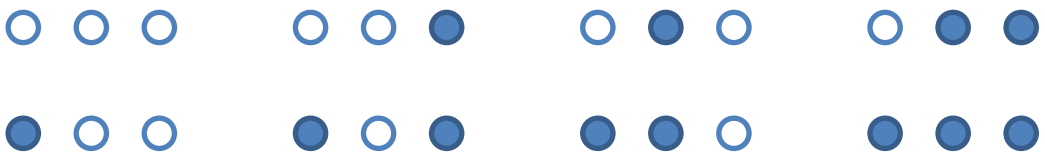
  →制限時間オーバー.



問6：微生物発電

解法

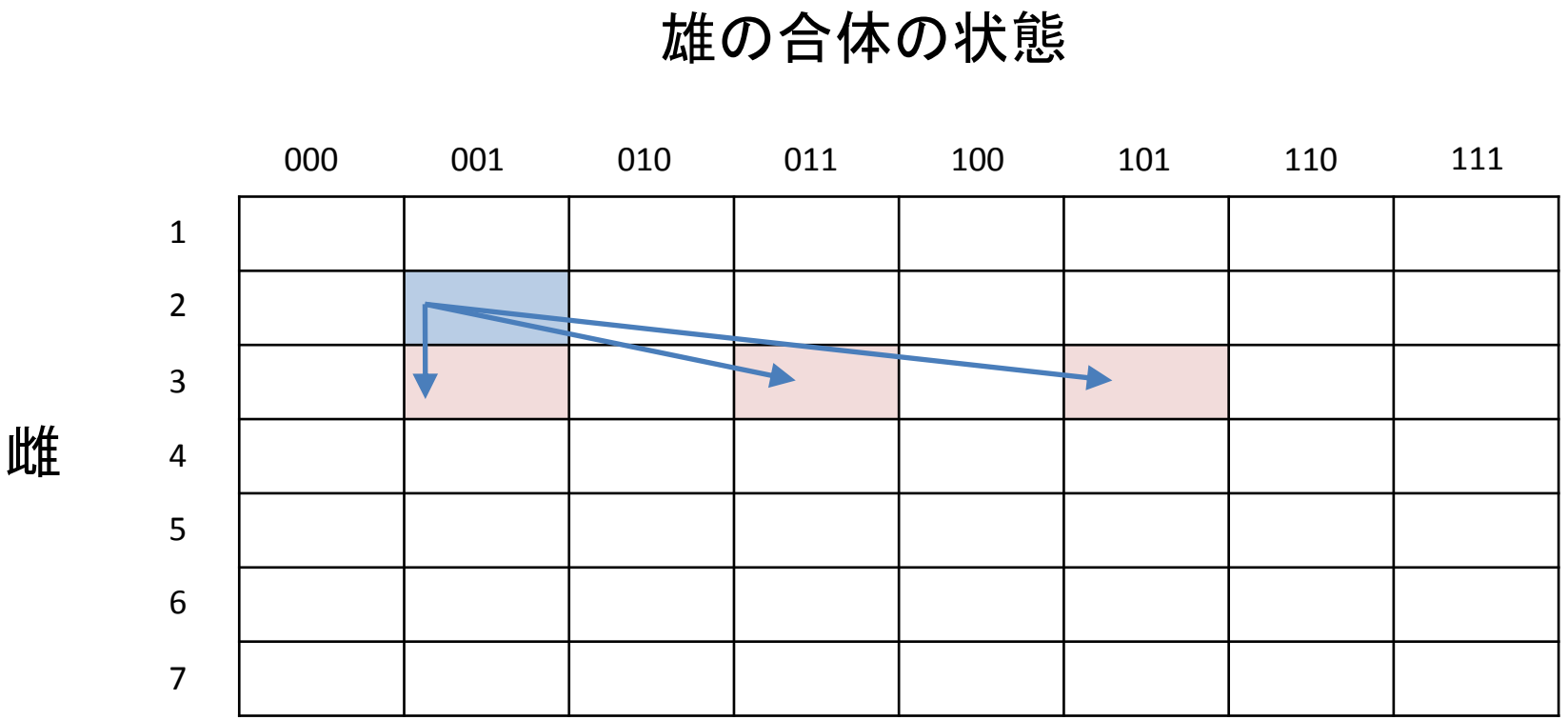
- 雄（または雌）の合体の状態をビットで保持.
  - 2^{12}通り程度
  - 例：Mが3のときは8通り



- i 番目までの雌を使って「雄の合体の状態」になったときのエネルギーの最大値を動的計画法で計算.

問6：微生物発電

解法



動的計画法で最大値を更新していく

問6：微生物発電

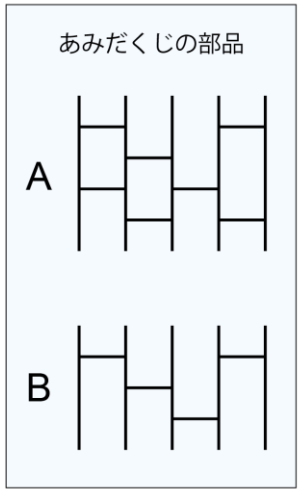
C++による解答例

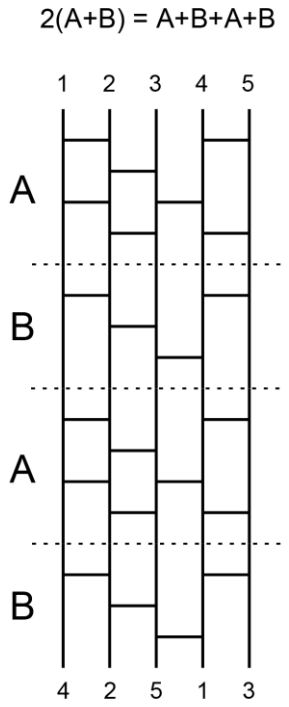
```
dp[0][0] = 0;

for ( int i = 0; i < M; i++ ){
  for ( int k = 0; k < (1<<W); k++ ){
    if ( dp[i][k] == -1 ) continue;
    dp[i+1][k] = max(dp[i+1][k], dp[i][k]);
    for ( int j = 0; j < W; j++ ){
      if ( k & (1<<j) ) continue;
      int ni = i+1;
      int nj = (k | (1<<j));
      dp[ni][nj] = max(dp[ni][nj], dp[i][k] + f(bm, bw, i, j));
    }
  }
}
```

問7：古代遺跡の謎

問題概要

- あみだくじの部品と、部品の連結を表す式が与えられる.
- 初期状態としてあみだくじの上部に1, 2, … N の数が順番に振られている.
- 式の通りにシミュレーションした後の数の並びを表示するプログラムを作成しなさい.
- 部品や部品の列の繰り返しを含
  - 例： 1000A+9999999(3X+2Y)
- 繰り返しの回数はとても多い場合がある.





問7:古代遺跡の謎

解法

- 与えられた式を構文解析で読み取る.
- 変換の繰り返しは
  - 繰り返し自乗法、あるいは
  - 変換を繰り返すと元に戻るので、周期を求め、繰り返す数を周期で剰算.

問7：古代遺跡の謎

解法１：繰り返し自乗法

- 部品を置換行列に変換.
- 現在の列を行列で表現し、置換行列を乗算することで部品による変換後の列を求める.
- 繰り返し自乗法を使用することで高速に変換後の列を求める.

問7：古代遺跡の謎

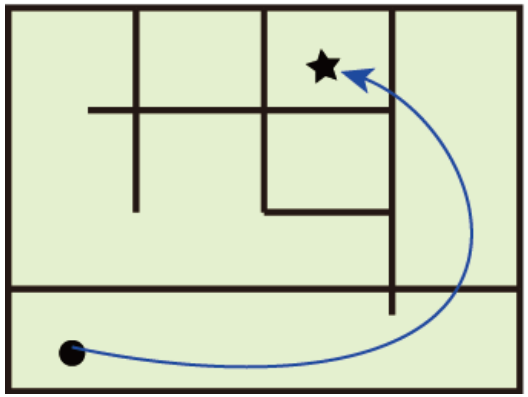
解法２：周期性を利用

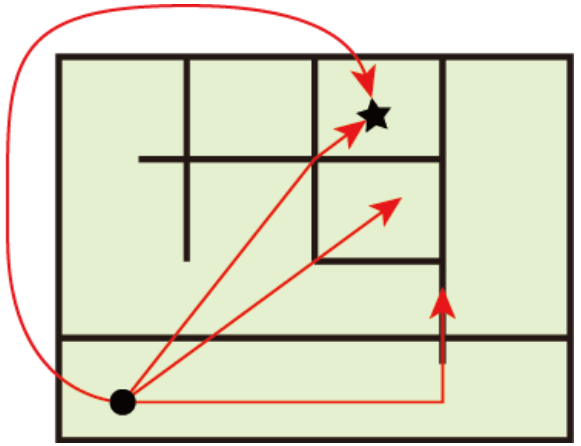
- 置換には周期性がある.
- 縦棒の数を表す整数Nの値が小さいため、何回か置換を繰り返すことで周期を求めることができる.
- 繰り返し回数をこの周期で剰算を取ることで小さくすることができる.

問8：壁

問題概要

- W*Hの長方形の領域がある.
- 領域内に水平または垂直な線分がいくつかある.
- 領域内のある2点が与えられるので、2点を端点とし、領域の長方形や線分の交点と交差しない曲線のうち、交差する線分の最小数を答えよ.





問8：壁

解法

- 線分が水平または垂直で、100個しかない.
  →座標圧縮して2次元グリッドに落とす.
  （最大で約400*400ぐらい）
- 線分で囲われた各領域で深さ優先探索（DFS）で塗りつぶして区別する.
- 各領域を頂点とし、隣り合った領域同士にコスト1の辺を張ったグラフを作る.
- このグラフで幅優先探索（BFS）を行い最短距離を求める.

問9 アルゴリズム検定試験

問題概要

- ある試験を運営する最小の費用を求めたい.
- 入力:
  - 受験者の座標.
  - 会場の費用、収容人数の上限、座標.
  - 単位距離あたりのバス運営費用.
- 出力:
  - 受験者が受験可能になるための最小の費用.
- 最適化するもの:
  - バスを走らせる距離D.
  - それぞれの会場について使用するかどうか.
  - 人と会場の割り当て.

問9 アルゴリズム検定試験

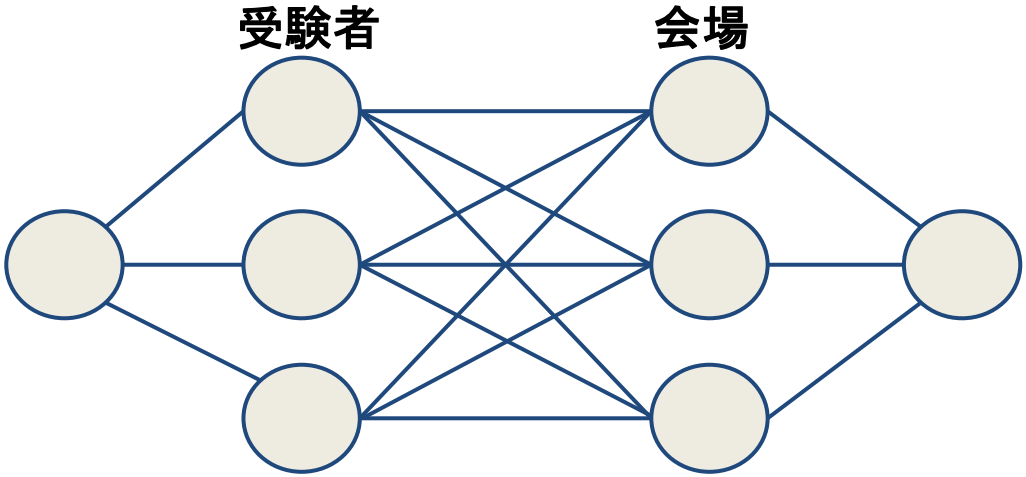
解法：問題を分割、会場について考える

- 使用する会場を決めると、会場の費用は一定
- 会場は最大5個
- 小さいので全探索： $2^5 = 32$

問9 アルゴリズム検定試験

解法:割り当てについて考える

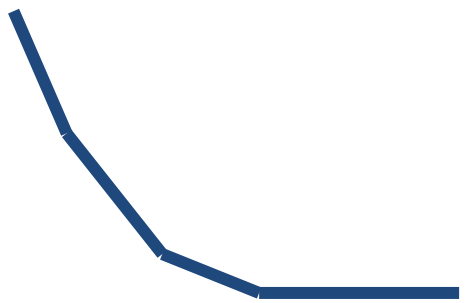
- バスを走らせないとすると・・・最小費用流
  - 受験者が会場に行く費用の合計を最小化.
  - 会場はそれぞれ収容制限がある.
  - 受験者はちょうど一つの会場に行かなければならない.



問9 アルゴリズム検定試験

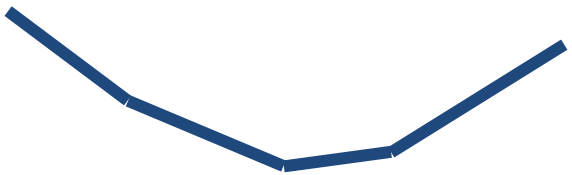
解法：バスの運営距離ついて考える

- 割り当て: Dについて、補助金の傾きが非減少
  - 重要な制約 ： $F(i+2)-F(i+1) \geqq F(i+1)-F(i)$
  - F(i):= 使用する会場を固定したとき、割り当てのうち移動補助金が最小となる金額
- バスの費用: Dについて、傾きが一定



移動補助金

バス費用

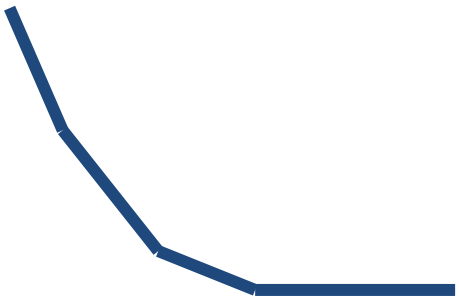


合計

問9 アルゴリズム検定試験

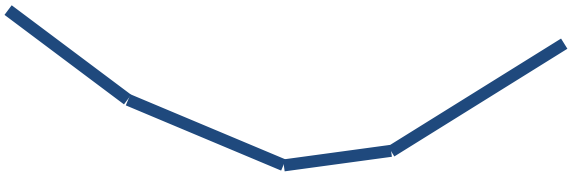
解法：Dを決定する

- 傾きが非減少
  - 傾きが負になる最大のDを二分法で探す.
- 下に凸
  - 極値を三分探索で探す.
- 二分法、三分法どちらでも適用可能



移動補助金

バス費用



合計

問9 アルゴリズム検定試験

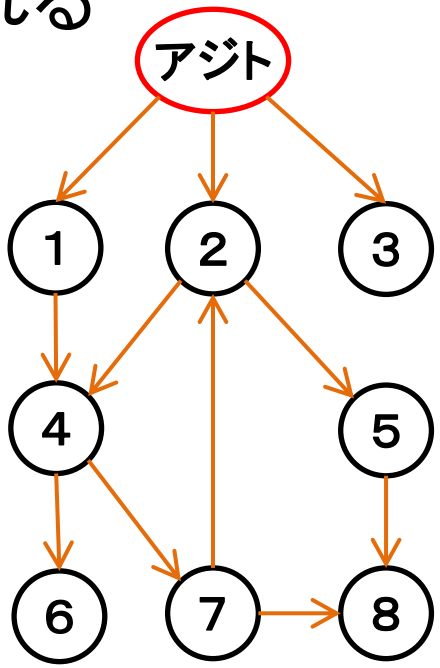
計算量

- 使用する会場を決める
  - 2^m
- Dを決め打ちし、二分法する
  - $\log_2 D$
- 最小費用流で最小の割り当て時の解を得る
  - $N \times (N + M) \times \log_2(N \times M)$
- 計算量
  - 1ケース3000万程度

問題10　アカ･ベコ捕獲作戦

状況:

- 節点(地点)と有向辺(道)のグラフが与えられる
- 道は一方通行
- ループはありえる(2→4→7→2 →･･)



求めるもの:

- アジトから受渡し場所までに必ず通る地点
- 受け渡し場所までの間に経由する地点の数が最も少ない地点
- 必ず通る地点が受渡し場所しかない場合は受渡し場所

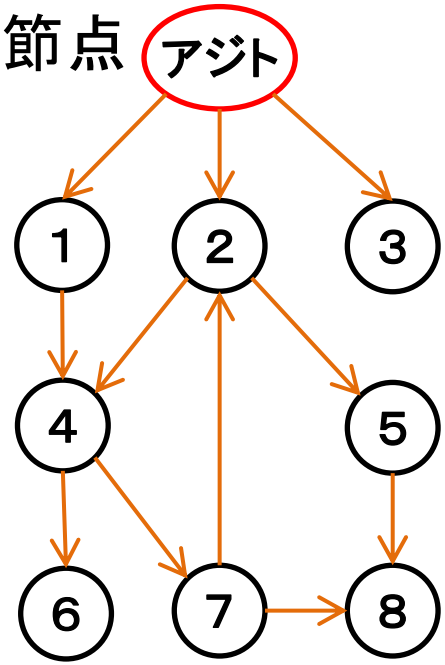
直接支配節による解法

- 根rをもつ有向グラフの、節点w(≠r) の直接支配節 idom(w) とは
  - 根rからwまでに必ず通る、w以外の節点
  - wまでの間に経由する節点の数が最も少ない節点

例）右のグラフの各節点の直接支配節

節点w	1	2	3	4	5	6	7	8
idom(w)	ア	ア	ア	ア	2	4	4	ア

（アはアジト）

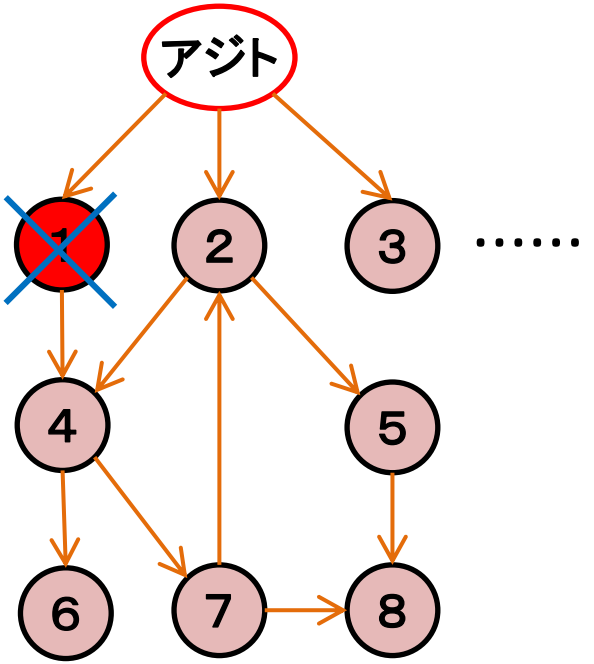


- アジトを有向グラフの根としたとき、求めるものは
  - 受渡し場所の直接支配節がアジトでないときは直接支配節
  - それ以外なら受け渡し場所
- 最大サイズは節点N=100,000、辺M=300,000
  → O(MN)よりは効率よく直接支配節を求める必要がある。

ある節点に到達するとき必ず通る地点を求める方法　O(NM)

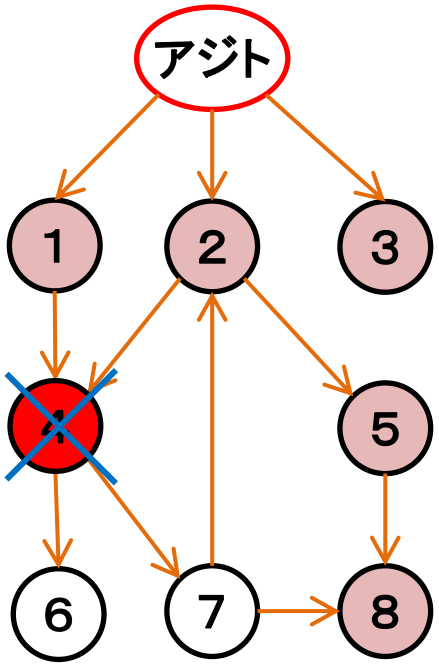
必ず通る節点 ⇒ そこを封鎖すると到達できない節点 ⇒
封鎖して到達できない節点は封鎖した節点を必ず通る必要がある。

1を封鎖→どの節点にも到達可能。



どの節点を受渡し場所にしても、
1を通らない経路がある。

4を封鎖→アジトから6や7へ到達できない。



6か7が受渡し場所の
とき、4は必ず通る。

受渡し場所wに到達するときに必ず通る節点が、idom(w)の候補。

候補の中で、そこから受渡し場所wまで他の候補を通らずに到達できるものがidom(w)。

効率よく解く方法　O(M logN)

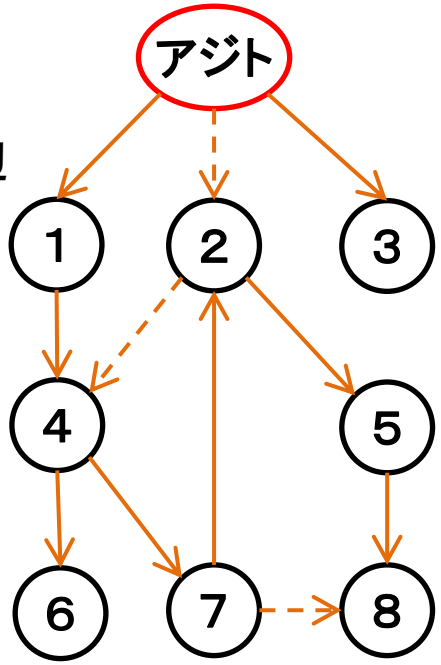
- 節点wの直接支配節は、アジトを根とする全域木Tの、根からwまでの木に沿った経路P_w上にある。
- ただし、P_wから外れた後にwでP_wに合流する経路があるとき、その経路により迂回されるP_w上の節点はidom(w)ではない。

（P_wから外れた後wで合流する経路がなければ、wのTでの親が idom(w) ）

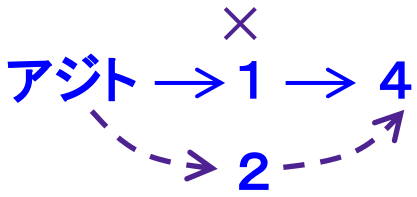
→ 木T上の有向辺

⇢ それ以外の有向辺

実線の辺だけ残すと、
全域木Tが得られる。



実線がアジトから4までのT上の経路P_4



アジトでP_4から外れて4で合流
⇒節点1は idom(4)ではない。

節点4の直接支配節はアジト

半支配節

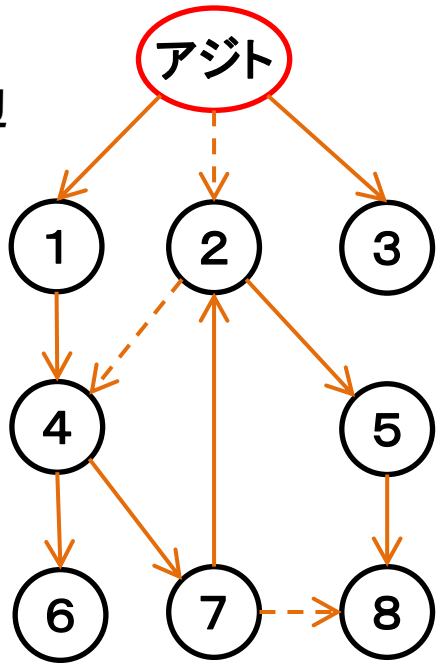
- 各節点wのidom(w)を求めるには、P_wから外れた後にwでP_wに合流する経路の始点を求めることが重要。
- そのような節点のうち、（全域木T上の辺だけを考えたときに）最も根に近いものをwの半支配節sdom(w) と呼ぶ。

（P_wから外れてwで合流する経路がなければ、wのTでの親がsdom(w) ）

→ 木T上の有向辺

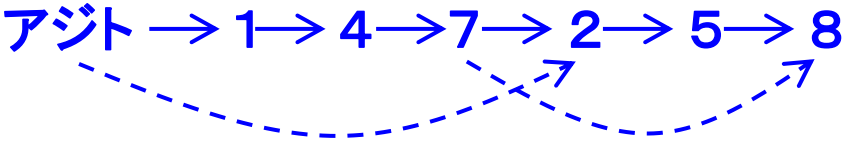
⇢ それ以外の有向辺

実線の辺だけ残すと、
全域木Tが得られる。



節点w	1	2	3	4	5	6	7	8
sdom(w)	ア	ア	ア	ア	2	4	4	7
idom(w)	ア	ア	ア	ア	2	4	4	ア

実線がアジトから8までのT上の経路P_8



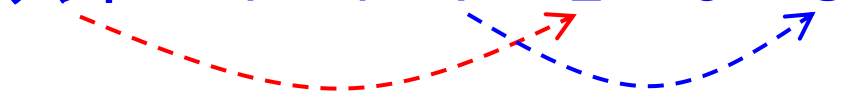
P_8から外れて8で合流する経路の始点は7。

節点8の半支配節は7

半支配節から直接支配節を求める方法

- 根からsdom(w)の手前までの節点でP_wから外れ、sdom(w)からwまでの間の節点（sdom(w)とw以外）で合流する経路があるときは、sdom(w)はwの直接支配節ではない。

実線がアジトから8までのT上の経路P_8



節点w	1	2	3	4	5	6	7	8
sdom(w)	ア	ア	ア	ア	2	4	4	7
idom(w)	ア	ア	ア	ア	2	4	4	ア

アジトでP_8から外れて2で合流する経路がある。
⇒8の半支配節7を通らずに8に行ける経路がある。

節点8の直接支配節はアジト

- 上記のような経路があるとき、その中で始点が最も根に近い経路のP_wとの合流点をuとすると、wの直接支配節はuの直接支配節。
- それ以外のときは、wの直接支配節はwの半支配節。

Lengauer-Tarjanのアルゴリズム O(MlogN)

- 以上をまとめたものが Lengauer-Tarjanのアルゴリズム。
コンパイラが目的プログラムの最適化を行うときなどに使われる。
  1. アジトを始点としてDFSを行い、アジトを根とする全域木Tを作る。
  2. 根以外の全節点について半支配節を求める。
     （経路圧縮を行うと、O(MlogN) で求まる。
     詳細は以下の原典を参照。）
  3. Tの先行順（preorder）で各節点を訪問し、半支配節から直接支配節を求める。

原典： A Fast Algorithm for Finding Dominators in a Flowgraph,
T. Lengauer and R. E.E Tarjan, ACM Transactions on Programming Languages and Systems, Vol. 1, No. 1, Pages 121-141 (July 1979).

参考図書： 最新コンパイラ構成技法（翔泳社, 2009年）19.2節

サンプルコード

```
for (int i=1; i<=nNode; ++i) semi[i] = 0;
   n = 0;   r = 1;
   DFS(r);
   label[0] = semi[0] = 0;

   for (int i=n; i>=2; --i) {
    int w = vertex[i];
    ITER(v,pred[w]) {
            int u = EVAL(*v);
            if (semi[u] < semi[w]) semi[w] = semi[u];
    }
    bucket[vertex[semi[w]]].push(w);
    LINK(parent[w], w);
    while (!bucket[parent[w]].empty()) {
            int v = bucket[parent[w]].front();
            bucket[parent[w]].pop();
            int u = EVAL(v);
            dom[v] = (semi[u] < semi[v]) ? u : parent[w];
    }
   }
```

```
for (int i=2; i<=n; ++i) {
  int w = vertex[i];
  if (dom[w] != vertex[semi[w]])
          dom[w] = dom[dom[w]];
}
dom[r] = 0;
```

DFS: 節点にDFS順に番号を付ける
EVAL: 半支配節を求める手順(原典参照)
LINK: 第2引数の先祖を第1引数にする

dom[i] が1: i番目の地点の答えはそれ自身
dom[i]が1以外: dom[i]の値が答え