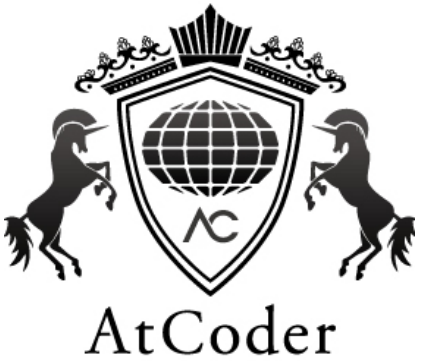
実践・最強最速のアルゴリズム勉強会
第四回　講義資料



AtCoder株式会社 代表取締役
高橋 直大

はじめに！

AtCoder

- 本講義では、ソースコードを扱います。
- 前面の資料だけでは見えづらいかもしれないので、手元で閲覧できるようにしましょう。
- URLはこちらから
  - http://www.slideshare.net/chokudai/wap-atcoder3
  - URLが打ちづらい場合は、Twitter: @chokudaiの最新発言から飛べるようにしておきます。
    - フォローもしてね！！！

目次

1. 勉強会の流れ
2. bitDP
3. 計算量の削減の仕方
4. 本日のまとめ

勉強会の流れ

1. 勉強会の日程
2. 1日の流れ

勉強会の日程

AtCoder

1日目　シミュレーションと全探索

2日目　色々な全探索

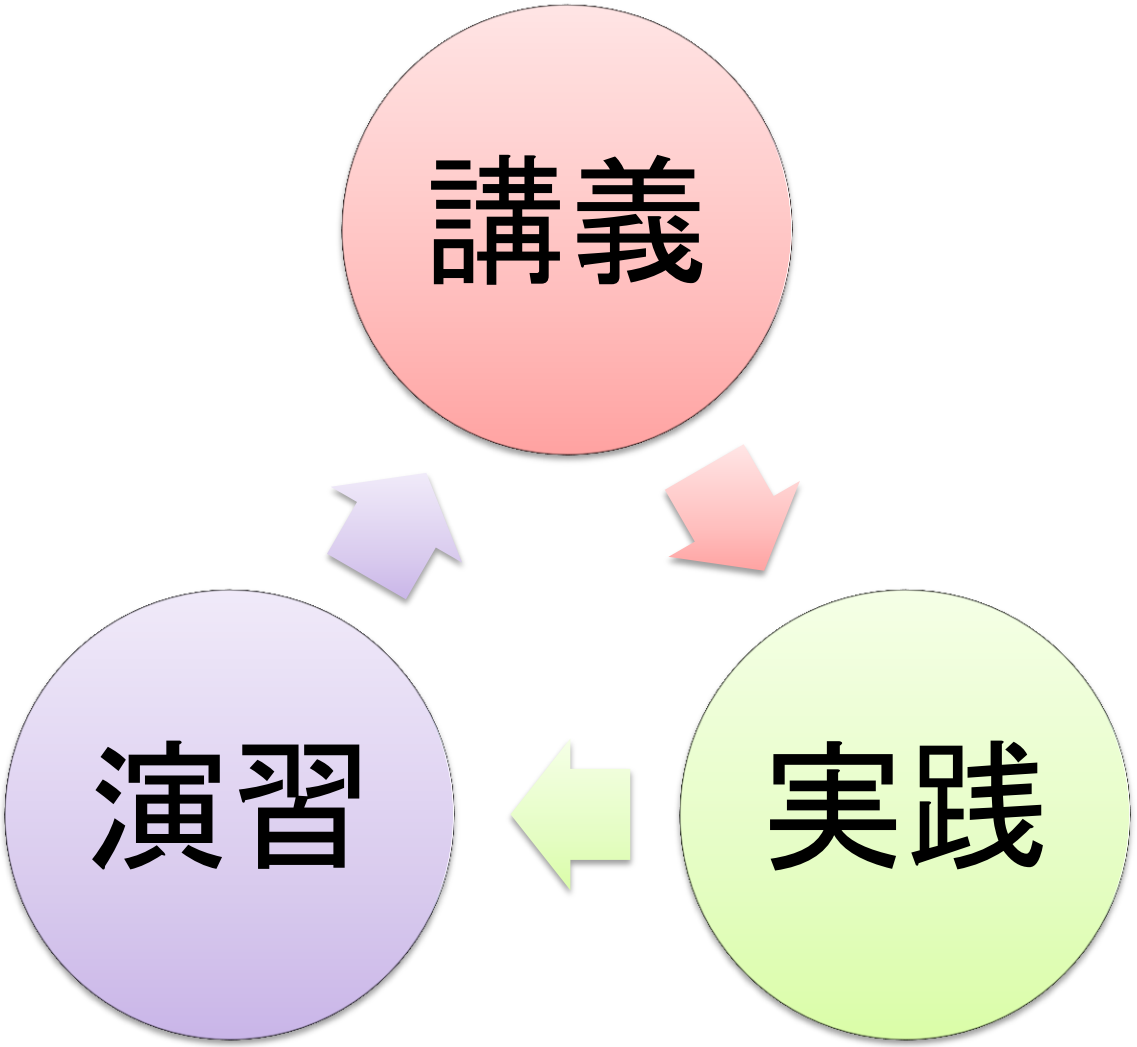
3日目　動的計画法とメモ化再帰

4日目　動的計画法と計算量を減らす工夫

5日目　難問に挑戦！

１日の流れ

AtCoder



１日の流れ

AtCoder

講義

- 基礎的なアルゴリズムを学ぶ
- 必要な知識を補う

実践

- 実際に問題例を見る
- コードでの表現方法を覚える

演習

- 自分で問題を解いてみる！
- コードを書いて理解を深める

１日の流れ

AtCoder

- 演習について
  - 実力差があると思うので、暇な時間が出来る人、ついていけない人、出ると思います。
    - どうしようもないので、早い人は支援に回ってもらえると嬉しいです！
  - 解らないことがあったら、#WAP_AtCoderでTwitterに投稿！
    - 多分早く終わった人が質問回答してくれます。
    - 具体例はこんな感じ
      - 「今やってる問題どれですか！」
      - 「この解答のWAが取れません！　http:// ~~」
      - 「コンパイルエラー出るよーなんでー＞＜　http:// ~~」
      - とりあえずコードが書けてたら、間違ってても提出してURLを貼りつけよう
  - もちろん、手を上げて質問してくれてもOK!
    - 回りきれる範囲では聞きに行きます。

今日の流れ

1. 前回までの復習
2. 今回やること

前回までの復習

AtCoder

- 全探索
  - 枝分かれの数が定数の場合 （第一回）
    - N重のforループによる全探索
  - 枝分かれの回数が不定の場合 （第二回）
    - 深さ優先探索
    - 幅優先探索
    - Bitを利用した深さ優先探索
      - 枝分かれが二股の時限定

前回までの復習

AtCoder

- 計算が間に合わない場合
  - 計算量の概念を用いた計算時間の予測（第３回）
  - メモ化再帰
    - 深さ優先探索をメモするようにしたもの
  - 動的計画法
    - メモ化再帰の向きを逆転し、forループのみで書けるように！

今日やること

AtCoder

- 動的計画法で扱える範囲を増やす！
  - 整数で集合を表す手法
  - 如何にして状態数を減らすか？
- 他にも色々ある、計算量を減らす工夫に触れる！
  - 貪欲法
  - 二分探索（時間があれば）

今日の講義

1. BitDP
2. その他の計算量を減らす工夫
3. 本日のまとめ

BIT DP

1.整数で集合を表す方法

2.Bit DP

Bitを利用した二分木の全探索　復習

AtCoder

- 問題
- OとXのみで構成されたN文字の文字列を列挙して、～～～～しなさい。
  - 深さ優先探索で行けるけど、再帰関数書くの大変・・・。

- 再帰関数を書かなくても、for文だけで書けてしまう！

Bitを利用した二分木の全探索　復習

AtCoder

- 例えば、OとXのみで構成された5文字の文字列の全列挙をする時
  - for(int i=0; i<32;i++)
  - 実は、このfor文で全列挙出来てしまう

Bitを利用した二分木の全探索　復習

AtCoder

- 整数を2進数で表す
  - 0 … 00000
  - 1 … 00001
  - 2 … 00010
  - 3 … 00011
  - 4 … 00100
  - …..
  - 31 … 11111
- 0から31までの数字はこんな感じ

Bitを利用した二分木の全探索　復習

AtCoder

- 整数を2進数で表す
  - 0 … 00000 … OOOOO
  - 1 … 00001 … OOOOX
  - 2 … 00010 … OOOXO
  - 3 … 00011 … OOOXX
  - 4 … 00100 … OOXOO
  - …..
  - 31 … 11111 … XXXXX
- 0から31までの数字はこんな感じ
  - 0,1を、O,Xに対応させてしまえば良い！

Bitを利用した二分木の全探索　復習

AtCoder

- 整数のk桁目のbitを、k番目の分岐に対する選択と解釈することにより、forループで処理可能になる！
  - ○×ゲームの解答を10回行った結果の全列挙
    - k番目のbitが0ならk問目は○、そうでなければ×
  - 香車の進み方の全列挙
    - k番目のbitが0ならkマス目には止まらない。そうでなければ止まる

- 他にも、整数のbitで表せるものはたくさんある！
  - 先ほどの手紙問題だと、「誰に手紙を渡したか」を整数1つで持つことも可能

Bitを利用した二分木の全探索　復習

AtCoder

- 具体的な実装例

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int N = cin.nextInt();

    for(int i=0;i<(1<<N);i++){
        String st = "";
        for(int j=0;j<N;j++){
            if((i>>j)%2==0) st += "o";
            else st+= "x";
        }
        System.out.println(st);
    }
}
```

Bitを利用した二分木の全探索　復習

AtCoder

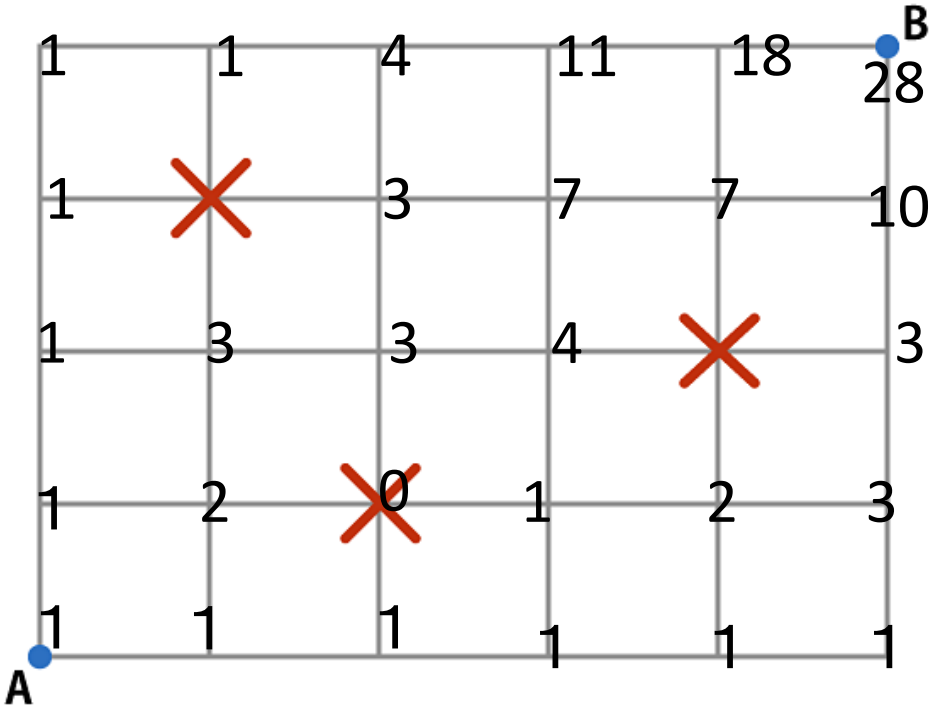
- 具体的な実装例
  - (1 << N)
    - 2のN乗。「2択がN回ある場合」の要素数。
  - ((i >> j) % 2)
    - iに対する、j番目のbit情報を取り出す。
      - » jは0から数えます。0-indexedです。
    - 例えば、i -> 12(= 1100)なら、
      - (12 >> 1) -> 6(110)
      - (12 >> 2) -> 3(11)
      - (12 >> 3) -> 1(1)
    - と、こんな感じで、右からj番目の要素を取り出せる

整数で集合を表す方法

AtCoder

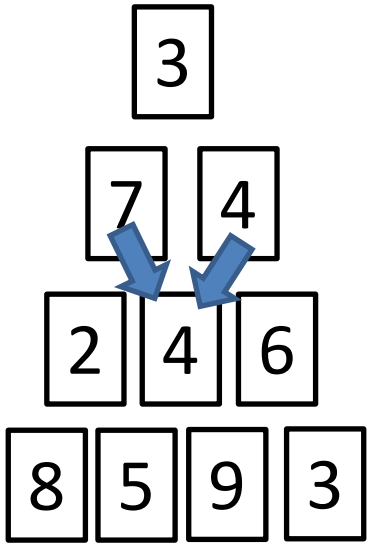
- 整数のbitを利用した、整数で集合を表す方法
  - 全探索の回でも使用しました！
  - 例えば、2進数で01011010のような場合
    - 右から見て1つ目のbitは0 → 使用しない
    - 右から見て2つ目のbitは1 → 使用する
    - というような、整数であるものに対する「集合」を表すことが出来た
- これを、動的計画法に活用しよう！

- 動的計画法や、メモ化再帰においては、「今の状態」を簡潔に書くことで、纏めてあげる必要があった。
  - ある地点に辿り着くまでの全パターンを纏める



- 動的計画法や、メモ化再帰においては、「今の状態」を簡潔に書くことで、纏めてあげる必要があった。
  - 水たまりを避けながら距離１，２，３のジャンプをする問題
  - その地点に到達するまでの、水たまりに入らなければならない最小の回数を纏める

- 動的計画法や、メモ化再帰においては、「今の状態」を簡潔に書くことで、纏めてあげる必要があった。
  - 上から辿って数字の大きいカードを獲得していく問題
  - そのカードに辿り着くまでの、数字の和の最大値を纏める

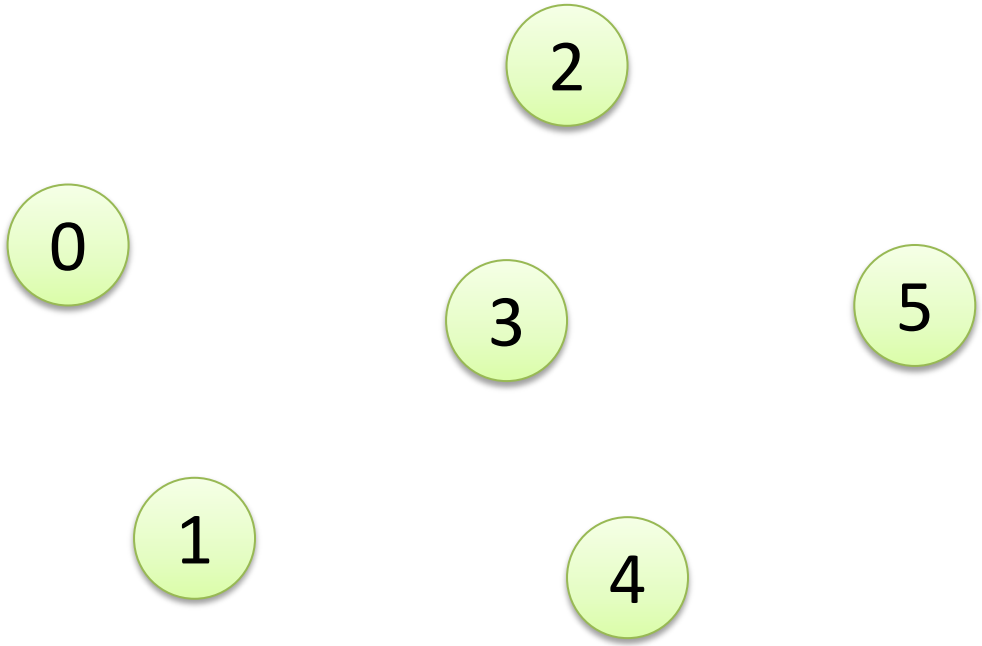


- 動的計画法において大切な事
  - それまでの状態を纏める！
    - 「ここまでの経路は違っても、ここから先は同じなもの」を纏めてあげることで、計算を省略できる！
    - 具体例は以下のようなもの
      - 組み合わせの個数なら、「そこまでの個数が何パターンあるか」
      - 最小値なら、「そこに辿り着くまでの最小の値」
      - 最大値なら、「そこに辿り着くまでの最大の値」
  - 少ない状態数に纏められるのであれば、動的計画法は無敵！
    - であれば、纏める手法をもっと覚えるべき！

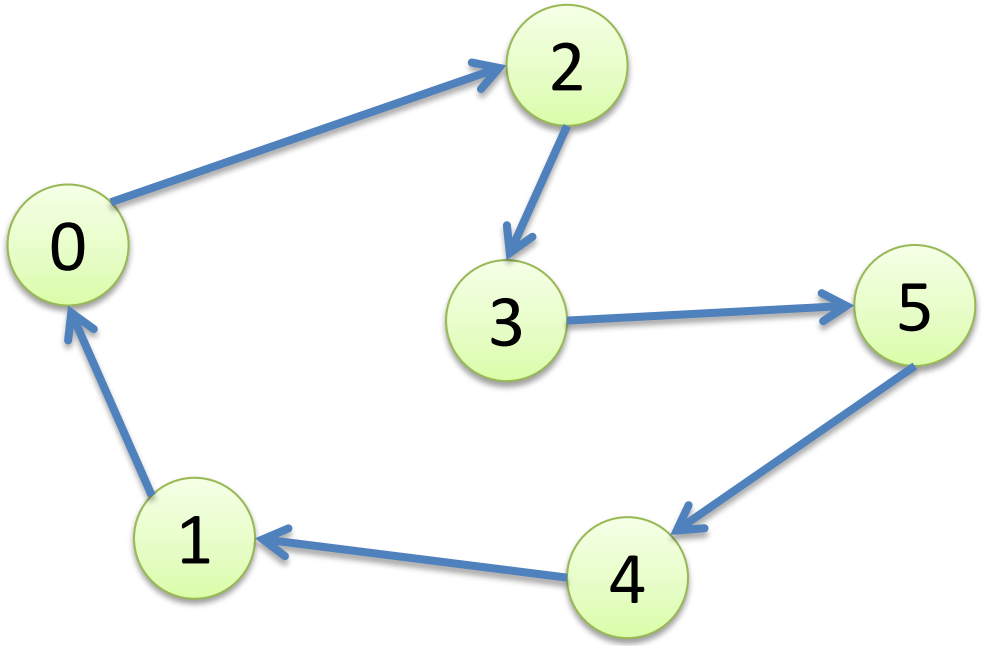
- 状態に、Yes,Noを大量に含むような問題
  - 結構色々な問題が存在する
    - 具体例１
    - 具体例２
    - 具体例３
    - 第2回講義に出た人なら、手紙を配る問題
  - これも上手くまとめてあげよう！

- 具体例　巡回セールスマン問題
  - 街がN個あります。街0から全ての街を通って帰ってきたいです。最短となる経路の長さを求めなさい。
  - 超有名問題
  - いわゆる、NP完全問題、と呼ばれるジャンル
    - 多項式時間(O(N^3)みたいなの)で解けない。
  - でも、Nが小さければ解ける！

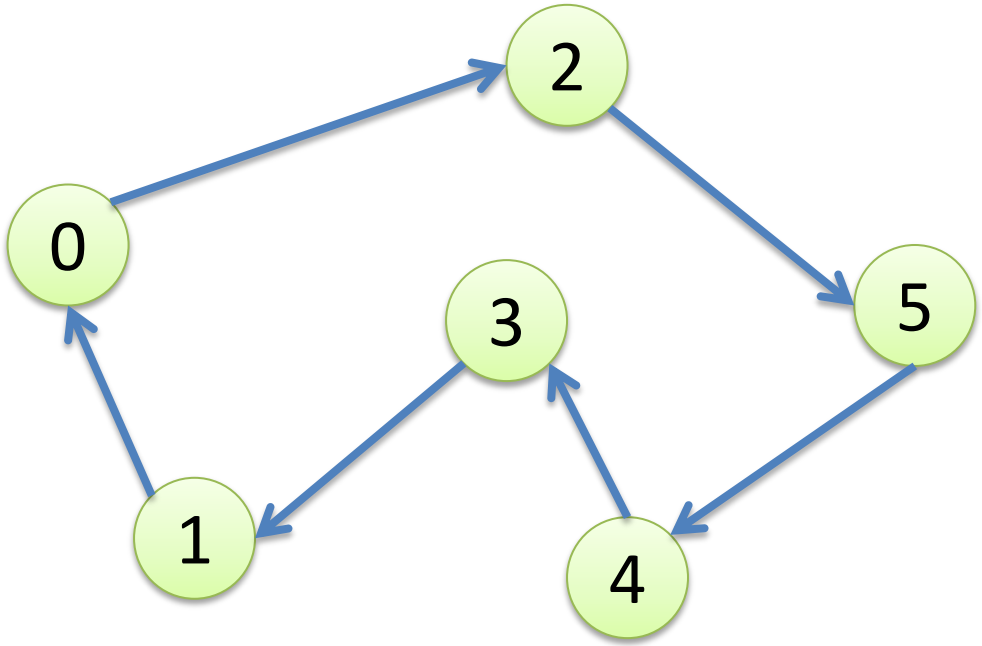
- 問題イメージ
  - N=6の時



- 問題イメージ
  - N=6の時　辿り方の一例

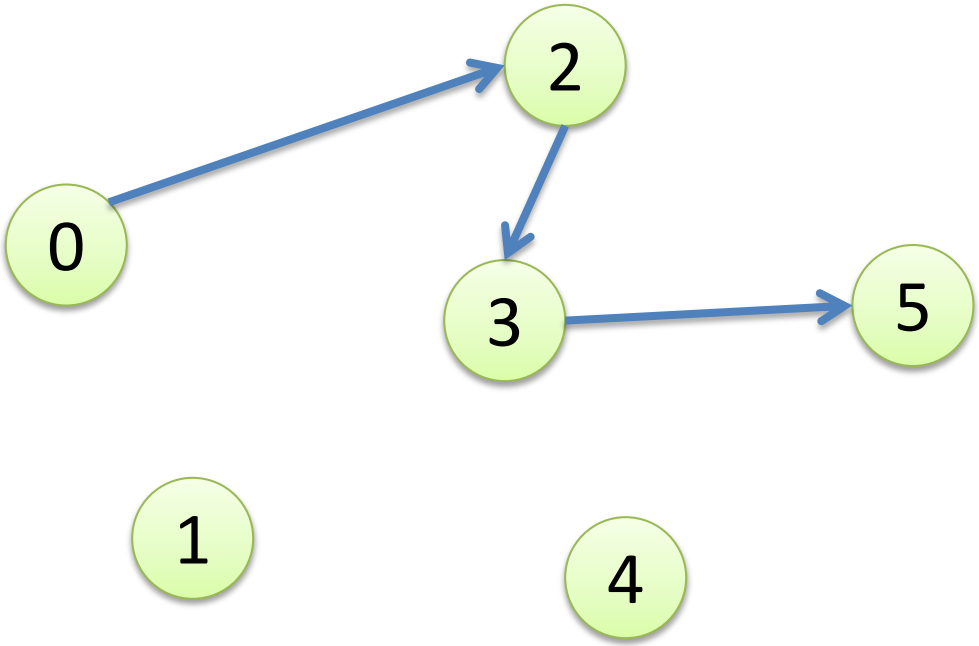


- 問題イメージ
  - N=6の時　辿り方の一例　辿り方は無数にある！

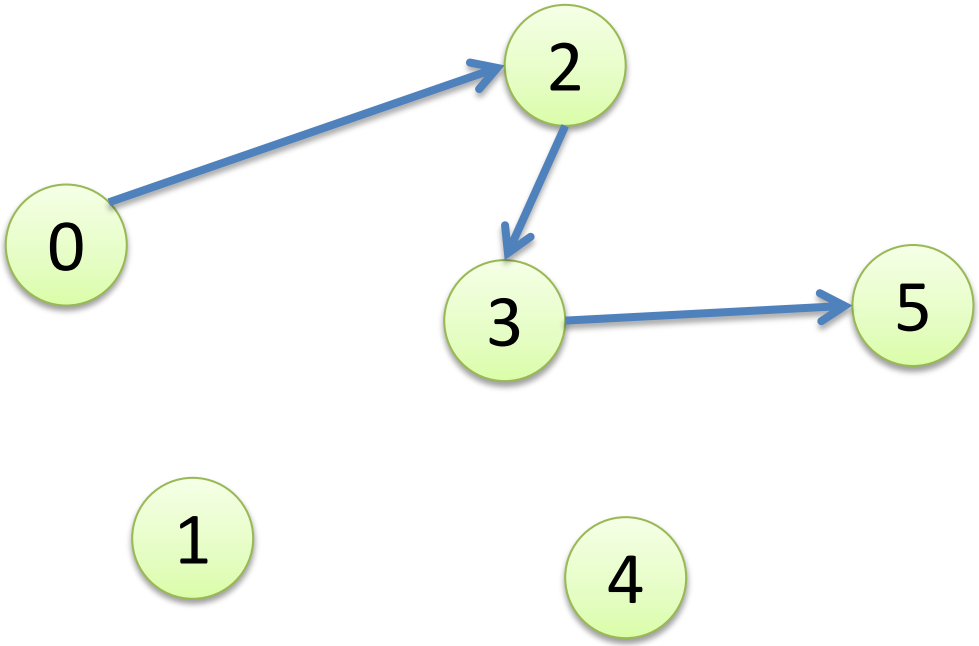


- 普通に全探索をするには？
  - N個街があるが、最初の街は0番
    - よって、選択肢はN-1個
    - 次の街は、まだ選んでない残りのN-2個
    - その次は、N-3個
    - ・・・
    - 最終的に、(N-1)!通りの辿り方が存在する。
  - 順列に対する全探索？
    - Nが8程度まではこれで解くことが可能
    - 今回は、Nが15程度まで解くことを目指します

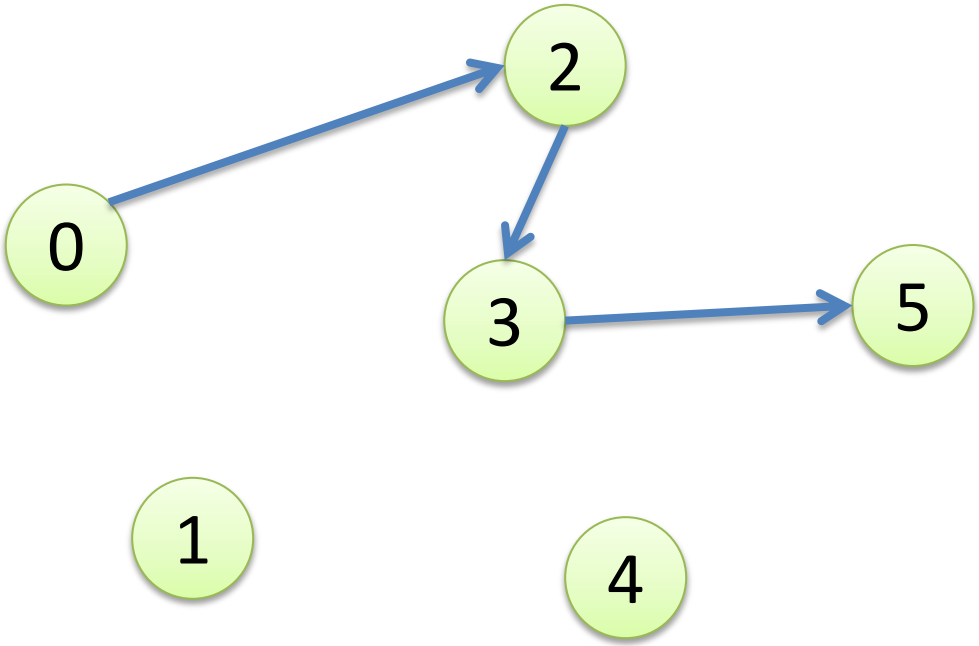
- 巡回セールスマン問題に対する状態って何だろう？
  - 例えば、ここまで調べた場合



- 巡回セールスマン問題に対する状態って何だろう？
  - 例えば、ここまで調べた場合
    - 辿ってきた経路をメモする？　例：{0, 2, 3, 5}と配列で持つ
    - 全組み合わせでバラバラになってしまうから全く纏まらない！



- 巡回セールスマン問題に対する状態って何だろう？
  - 例えば、ここまで調べた場合
    - 「ここから先が完全に一緒で、ここまでの経路が違うもの」を纏めたい！



- 巡回セールスマン問題に対する状態って何だろう？
  - {0, 2, 3, 5}と、この先の状態が同じになるのってどういうのだろう？
    - {0, 3, 2, 5}とか、残りが1,4で、最後に0に帰ってくるだけだから、ここから先は同じ結果が得られそう。
      - つまり、{0,2,3,5}, {0,3,2,5}の、どちらか片方の良い結果だけ、今後考えていけば良い。
    - 共通しているものは？
      - ここまでに通ってきた街の集合
      - 最後に訪れた街
    - この2つを状態として、上手いこと纏められないだろうか？

- 共通しているものは？
  - 最後に訪れた街
    - これはただの街の番号を覚えておけば良い
  - ここまでに通ってきた街の集合
    - 0,2,3,5に通って、1,4がまだ、という情報を保持したい。
      - 2進数を使用して、整数で保持すれば良い！

- 情報の保持のしかた
  - 0,2,3,5には訪れた
    - 0,2,3,5番目のbitを1にする！
  - 1,4には訪れていない
    - 1,4番目のbitを0にする！
  - つまりこの集合を表す整数は、2進数で101101
    - 後ろから読むことに注意
    - 実際は、0は必ず訪れているので、最後を削って10110で良い
  - こんな感じで集合を表してあげると、計算量がぐっと落とせる！

- 計算量について
  - 普通の全探索
    - 辿り方がN!通り
    - 経路の長さがN
    - 全体の計算量は、O(N! * N)
      - 8くらいまでが限界
  - 動的計画法を使った場合
    - 最後に訪れた街の種類が最大N通り
    - 街の状態数は2^N通り
    - 各状態に対して、最大N通りの次の街の選択肢
    - 全体の計算量は、O(2^N * N^2)
      - 15くらいまでいける

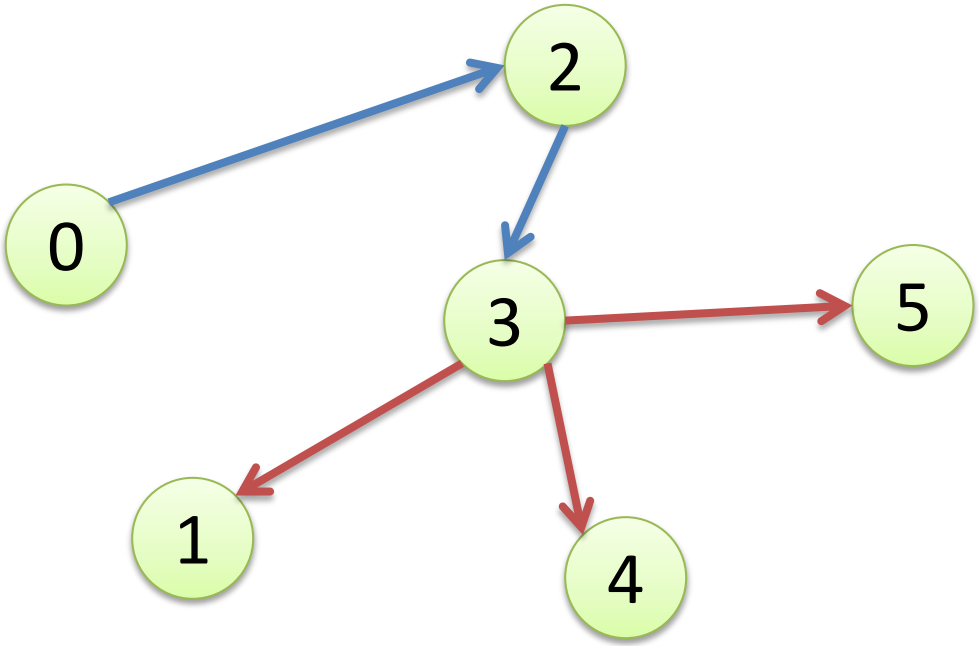
- ソースコード　入力処理

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int N = cin.nextInt();

    double[] x = new double[N];
    double[] y = new double[N];
    for(int i=0;i<N;i++){
        x[i] = cin.nextDouble();
        y[i] = cin.nextDouble();
    }

    double[][] dist = new double[N][N];
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            dist[i][j] = Math.hypot(x[i]- x[j], y[i] - y[j]);
        }
    }
```

- ソースコード　配列の初期化
  - DPの状態は、「訪れた街の集合」「最後に訪れた街」の順

```
//初期化、全ての状態に、大きな数INFを入れておく
double INF = 1e100;
double[][] dp = new double[1<<N][N];
for(int i=0;i<(1<<N);i++){
    for(int j=0;j<N;j++){
        dp[i][j] = INF;
    }
}
//初期状態、0に訪れていて、それ以外にはまだ訪れていない
dp[1][0] = 0;
```

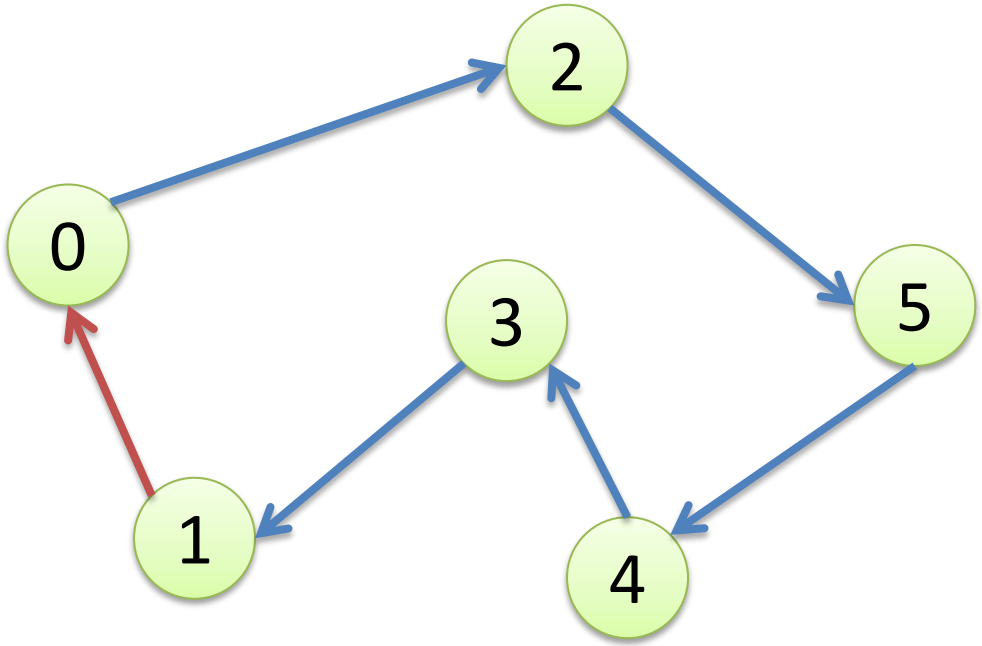
- 配るDPのイメージ
  - 0,2,3に対して回って、最後が3の時
  - ここから、残った全頂点に対して、配るDPを行う



- ソースコード　配るDP

```
for(int i=0;i<(1<<N);i++){
    for(int j=0;j<N;j++){
        if(dp[i][j] == INF) continue;
        //次の状態について探索する。配るDP
        for(int k=0;k<N;k++){
            //もしkが既に訪れた街であれば次へ
            if((i>>k)%2==1) continue;
            //次の集合に、今の集合からkbit目を立てる
            int nexti = i | (1<<k);
            //距離はdist[j][k]で得られるので、今までの分に足す
            double nextd = dp[i][j] + dist[j][k];
            //次の場所は、もちろんk
            //今まで調べたものより短くなっていれば、更新する
            dp[nexti][k] = Math.min(dp[nexti][k], nextd);
        }
    }
}
```

- 最後の処理
  - 全てを回りきっても、最後に戻る処理があるのを忘れず



- ソースコード　出力部
  - 全て回ったパターンに対し全探索
  - 最後に0に帰る処理があるのにも注意

```
//全部辿り着いた状態allを定義しておく
int all = (1<<N) - 1;
double ret = INF;
for(int i=0;i<N;i++){
    //最後の街がiであることがあり得ないならcontinue
    //今回の場合は、0を含めているので、0の場合引っかかる
    if(dp[all][i] == INF) continue;
    //0に帰ってくるまでの経路を足す
    double temp = dp[all][i] + dist[i][0];

    ret = Math.min(ret, temp);
}
System.out.println(ret);
```

BitDP ソースコードコピペ用

AtCoder

```
import java.util.Scanner;

public class Main{
public static void main(String[] args){
new Main().run();
}

void run()
{
Scanner cin = new Scanner(System.in);
int N = cin.nextInt();

double[] x = new double[N];
double[] y = new double[N];
for(int i=0;i<N;i++){
x[i] = cin.nextDouble();
y[i] = cin.nextDouble();
}

double[][] dist = new double[N][N];
for(int i=0;i<N;i++){
for(int j=0;j<N;j++){
dist[i][j] = Math.hypot(x[i]- x[j], y[i] - y[j]);
}
}

//初期化、全ての状態に、大きな数INFを入れておく
double INF = 1e100;
double[][] dp = new double[1<<N][N];
for(int i=0;i<(1<<N);i++){
for(int j=0;j<N;j++){
dp[i][j] = INF;
}
}
//初期状態、0に訪れていて、それ以外にはまだ訪れていない
dp[1][0] = 0;

for(int i=0;i<(1<<N);i++){
for(int j=0;j<N;j++){
if(dp[i][j] == INF) continue;
//次の状態について探索する。配るDP
for(int k=0;k<N;k++){
//もしkが既に訪れた街であれば次へ
if((i>>k)%2==1) continue;
//次の集合に、今の集合からkbit目を立てる
int nexti = i | (1<<k);
//距離はdist[j][k]で得られるので、今までの分に足す
double nextd = dp[i][j] + dist[j][k];
//次の場所は、もちろんk
//今まで調べたものより短くなっていれば、更新する
dp[nexti][k] = Math.min(dp[nexti][k], nextd);
}
}
```

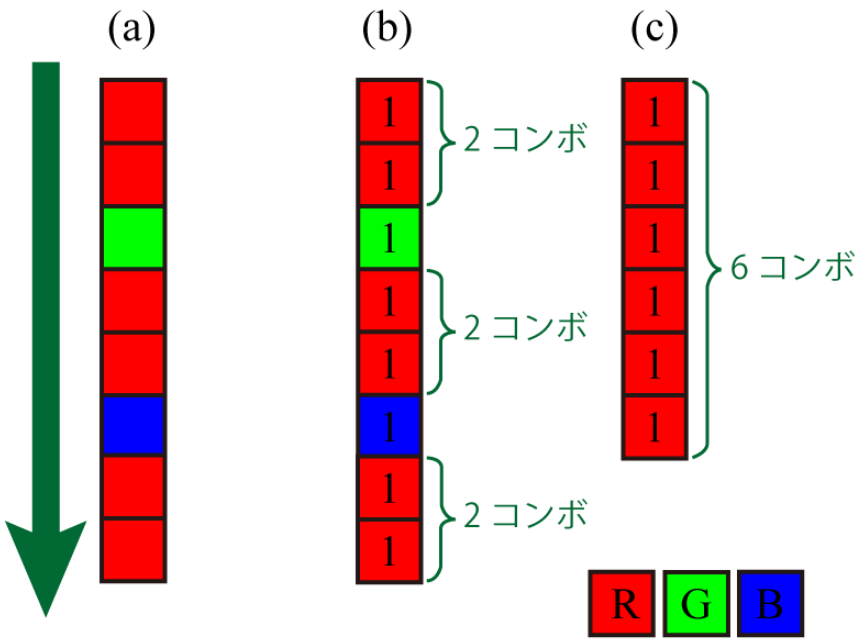
BitDP 実践

AtCoder

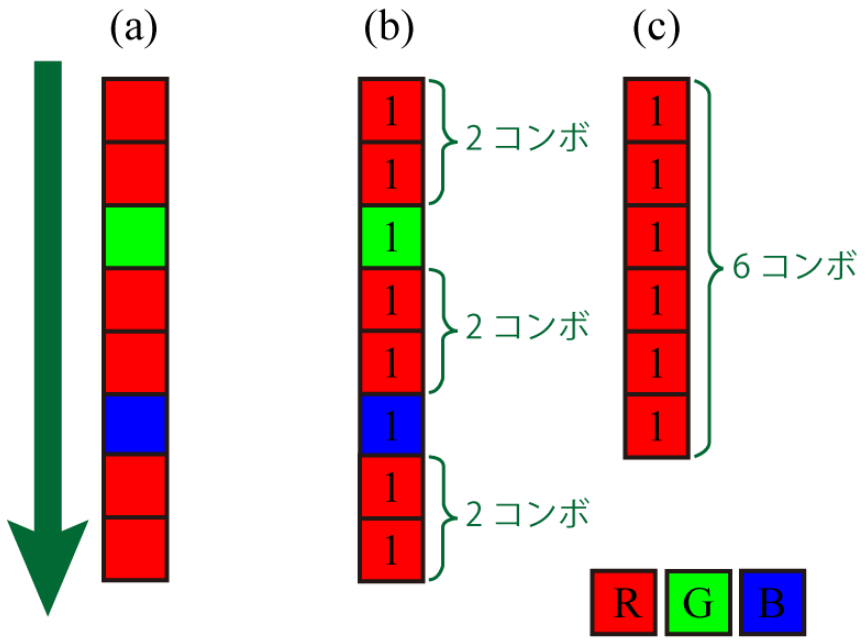
- ARC010 C問題　積み上げパズル
  - http://arc010.contest.atcoder.jp/tasks/arc010_3

- 問題概要
  - M色のブロックがn個、1個ずつ順番に降りてくる
  - 1つ1つのブロックに対して、「捨てる」「積み上げる」の2つの行動をとることが可能である
  - 最終的な状態によって、点数が決まる
    - 色ボーナス：色ごとに決められた得点が、山に含まれている個数分与えられます。
    - コンボボーナス：同じ色のブロックが x 個続いて積まれている場合、コンポボーナス配点 Y に応じて $Y \times (x-1)$ 点が与えられます。
    - 全色ボーナス：山の中に m 色のブロックがそれぞれ 1 個以上含まれていると Z 点が与えられます。
  - 得られる点数の最大値を求めなさい

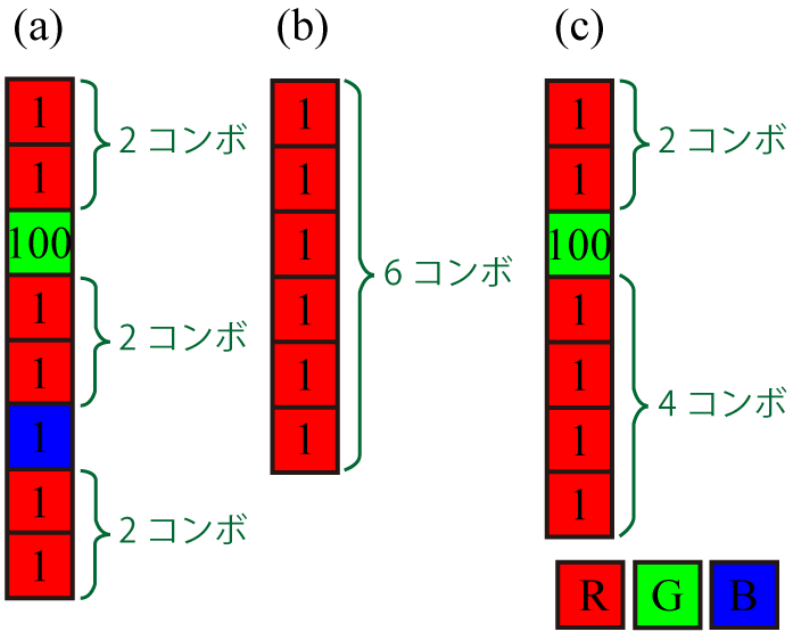
- N=8, M=3, Y=5, Z=3のとき、RRGRRBRR
  - (b)のように全て積み上げると、
    - 3個の2連コンボが各5点
    - 全色ボーナス3点
    - 全てのブロックが1点なので、色ボーナス8点
    - の26点が得られる



- N=8, M=3, Y=5, Z=3のとき、RRGRRBRR
  - (c)のように全て積み上げると、
    - 6連コンボが25点
    - 全てのブロックが1点なので、色ボーナス6点
    - の31点が得られる



- N=8, M=3, Y=5, Z=3のとき、RRGRRBRR
  - 配点が変わった時、(c)のように全て積み上げると、
    - 106(色ボーナス) + 5*(1+3)(コンボボーナス) = 126点
  - (b)は変わらず31点



- 解き方の方針
  - N<=5000, M<=10なので、全探索は出来ない。
    - 全て探索してしまうと、O(2^N)になってしまう
  - いくつかの法則に気付けば、状態を纏めることが出来る
    - コンボボーナスは、直前に置いた色と一致するたびにY点入る
      - 直前に置いた色さえ覚えておけば、途中計算可能！
    - 全色ボーナスは、「どの色を既に使ったか」を覚えておけば良い
      - これは、「これまで使った色」の集合を持って置けば良い
    - もちろん、「今いくつのブロックを使ったか」も必要
    - つまり、持つべき状態は、「今いくつのブロックを使ったか」「直前に置いた色」「どの色を使ったか」の3つ！

- ソースコード
  - http://arc010.contest.atcoder.jp/submissions/153201
  - 入力処理

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int n = cin.nextInt();
    int m = cin.nextInt();
    int Y = cin.nextInt();
    int Z = cin.nextInt();
    char[] c = new char[m];
    int[] p = new int[m];
    for(int i=0;i<m;i++){
        c[i] = cin.next().charAt(0);
        p[i] = cin.nextInt();
    }

    String st = cin.next();
```

- ソースコード
  - 初期化

```
int[][][] dp = new int[n+1][1<<m][m];
int INF = -99999999;
for(int i=0;i<=n;i++){
    for(int j=0;j<(1<<m);j++){
        for(int k=0; k<m;k++){
            dp[i][j][k] = INF;
        }
    }
}
dp[0][0][0] = 0;
```

- ソースコード
  - DP事前処理　何番目の色かを探す

```
//今注目しているブロックをtとする
for(int t=0;t<n;t++){
    //事前処理 何番目の色かを探す
    int num = 0;
    for(int i=0;i<m;i++){
        if(c[i] == st.charAt(t)){
            num = i;
            break;
        }
    }
```

- ソースコード
  - DP（配るDPによる実装）

```
    for(int i=0;i<(1<<m);i++){
        for(int j=0;j<m;j++){
            if(dp[t][i][j]==INF) continue;
            //捨てた時の処理
            dp[t+1][i][j] = Math.max(dp[t+1][i][j], dp[t][i][j]);

            //積み上げた時の処理
            int nexti = i | (1<< num);
            int nextpoint = dp[t][i][j];
            //色ボーナスを足す
            nextpoint += p[num];
            //コンボボーナスを足す 最初は足さないのに注意
            if(i!=0 && j==num) nextpoint += Y;

            dp[t+1][nexti][num]
                    = Math.max(dp[t+1][nexti][num], nextpoint);
        }
    }
}
```

- ソースコード
  - 出力処理

```
    int ret = 0;
    //全色ボーナスが発生しないパターン
    for(int i=0;i<(1<<m) - 1;i++){
        for(int j=0;j<m;j++){
            if(dp[n][i][j]==INF) continue;
            ret = Math.max(ret, dp[n][i][j]);
        }
    }

    //全色ボーナスが発生するパターン
    for(int j=0;j<m;j++){
        if(dp[n][(1<<m)-1][j]==INF) continue;
        ret = Math.max(ret, dp[n][(1<<m)-1][j] + Z);
    }

    System.out.println(ret);
}
```

- 解けてる？
  - 今回は解けたが、メモリ使用量がかなりギリギリ
    - メモリを削減する方法を覚えておくと便利。
  - ここで、減らせる部分を探す。
    - 配列の最初の項目「今いくつのブロックを使ったか」は、iとi+1の2つしかループ内で見ていない！
    - つまり、この2つだけを保持しておけば良い。

AtCoder

- 解き方
  - 「直前に置いた色」「今まで使った色の集合」の2つの要素に対して、「ここまでの最高得点」をメモする。
  - 「今どのブロックを見ているか」も本当は必要？
    - これから説明するテクニックでそれは解決することは可能！

BitDP実践

AtCoder

- 解き方
  - dp[1<<M][M]を用意する。
    - 今、集合Aを使って、最後にBを置いた時の最高得点がdp[A][B]である、ということを表す。
    - 今見ているブロックがiなら、先で言う3要素を状態としたDPで言うと、dp[i][1<<M][M]を表す。
  - nextdp[1<<M][M]を、ループ毎に用意する
    - 次のブロックを処理した後に、pから1つのブロックを置いた時の、最高得点を表す。
    - 今見ているブロックがiなら、先で言う3要素を状態としたDPで言うと、dp[i + 1][1<<M][M]を表す。
  - ループが終わるたびに、nextdpをdpにコピーする
    - こうすることで、メモリの最低限の要素しか見ないで良くなる

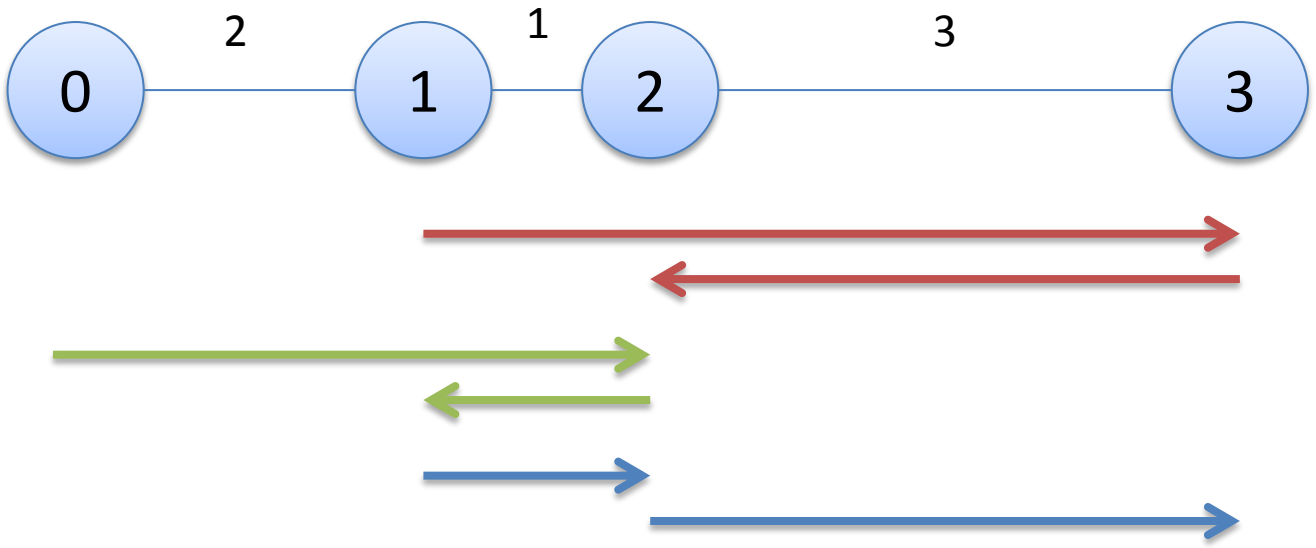
- 具体的なソースコード例

```
int[][] dp = new int[1<<m][m];
//今注目しているブロックをTとする
for(int t=0;t<n;t++){
    int[][] nextdp = new int[1<<m][m];

    //事前処理

    for(int i=0;i<(1<<m);i++){
        for(int j=0;j<m;j++){
            if(dp[i][j]==0) continue;
            //捨てた時の処理


            //積み上げた時の処理


        }
    }
    dp = nextdp;
}
```

- ARC020 D問題 お菓子の国の旅行
  - http://arc020.contest.atcoder.jp/tasks/arc020_4
    - もちろん部分点

- 問題概要
  - N個の街が一直線上に存在する
  - 街iと街i+1を繋ぐ道の長さはD_i
  - K個の街を選んで、その街を1回ずつ回りたい
  - 回り方がMの倍数であるパターンは何通りあるか？
    - パターンが多いので、1000000007で割った余りを求めよ
  - N≦12, M≦30, K≦min(N, 10)

- 問題の具体例
- N=4 M=4 K=3 D={2,1,3}の時、
  - 辿り方は6通り　下は例の3つ



- 解き方のヒント
  - さっきの問題と殆ど一緒！
  - 「今見ている場所」「最後にいる街」に加えて、「今までの距離をMで割った余り」を状態として持てば良い
    - 訪れた街の回数は、bitの立っている数で解る
  - これをもとに、今度は組み合わせの個数を求めてあげれば良い！

- 出来た人は以下の問題にも挑戦してみよう！
  - ARC016 C ソーシャルゲーム（難）
    - http://arc016.contest.atcoder.jp/tasks/arc016_3
  - ARC020 D お菓子の国の旅行（鬼）
    - http://arc020.contest.atcoder.jp/tasks/arc020_4

- ソースコード
  - http://arc020.contest.atcoder.jp/submissions/153196

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int N = cin.nextInt();
    int M = cin.nextInt();
    int K = cin.nextInt();
    int[] D = new int[N - 1];

    for(int i=0;i<N-1;i++){
        D[i] = cin.nextInt();
    }
    if(N>12) return;
```

- ソースコード

```
//2つの街の間の距離を計算する
int[][] dist = new int[N][N];
for(int i=0;i<N;i++){
    int nowdist = 0;
    for(int j=i+1;j<N;j++){
        nowdist += D[j-1];
        nowdist %= M;
        dist[i][j] = dist[j][i] = nowdist;
    }
}
```

- ソースコード

```
//配列の初期化
int mod = 1000000007;
long[][][] dp = new long[1<<N][N][M];
for(int i=0;i<N;i++){
    dp[1<<i][i][0] = 1;
}
```

- ソースコード

```
for(int i=0;i<(1<<N);i++){
    for(int j=0;j<N;j++){
        for(int k=0;k<M;k++){
            if(dp[i][j][k]==0) continue;
            //次の街を全通り調べる
            for(int l=0;l<N;l++){
                if((i>>l) % 2 == 1) continue;
                int nexti = i | (1 << l);
                int nextk = (k + dist[j][l]) % M;
                dp[nexti][l][nextk] += dp[i][j][k];
                dp[nexti][l][nextk] %= mod;
            }
        }
    }
}
```

AtCoder

- ソースコード

```
    //K個の街を訪れて、余りが0のパターンを全列挙
    long ret = 0;
    for(int i=0;i<(1<<N);i++){
        if(Integer.bitCount(i) != K) continue;
        for(int j=0;j<N;j++){
            ret += dp[i][j][0];
            ret %= mod;
        }
    }
    System.out.println(ret);
}
```

休憩！

計算量を減らす工夫

1.貪欲法
2.二分探索（時間があれば）

貪欲法

1.貪欲法って？
2.貪欲法を使える状況
3.貪欲法演習

貪欲法って？

AtCoder

- 貪欲法とは？
  - 選択肢が複数ある場合、普通は探索などを行って全てのパターンを調べる。
  - 貪欲法の場合は、適当な基準を用いて、最も良いものを決め打ちしてしまう！

貪欲法って？

AtCoder

- 問題例１
  - {2,3,4,5,6}の5つの数字があります。
  - ここから、3つの数字を選んで、積を求めます。
  - これを出来るだけ大きくしたい時、その答えを求めなさい。

貪欲法って？

AtCoder

- 探索だと・・・？
  - 3つの数字を全て全列挙する。
    - 3つのforループ？
- 貪欲法だと・・・？
  - 大きい順に選んでいく
    - 6, 5, 4の順に選んで、120と出力する！
    - この答えは正しい

貪欲法って？

AtCoder

- 問題例2
  - 1, 2, 4, 8, 16….のような、2の累乗の整数の和のみで、整数nを作りたい。
  - 使う数字の数を出来るだけ減らしたい時、使う数字の数はいくつになるか？

貪欲法って？

AtCoder

- 探索だと・・・？
  - 深さ優先探索？
    - Nから、1,2,4,8,16…を引いていくような深さ優先探索
    - メモ化も容易に可能！

- 貪欲法だと・・・？
  - 大きい順に足していく
    - 例えば43なら、最大の数である32を足す。残り17
    - それより小さい中で、最大である16を足す。残り1
    - 最後に1を足すので、答えは3個と解る。
    - この答えは正しい。

貪欲法って？

AtCoder

- 問題例3
  - 重さ100まで金塊を持つことが可能である。
    - 存在する金塊は、それぞれ重さ{13, 37, 50, 62}の4つである。
  - 出来るだけたくさんの金塊を持って帰りたい時、その重さはいくらになるか？

貪欲法って？

AtCoder

- 探索だと・・・？
  - 深さ優先探索？
    - 「持って帰る」「持って帰らない」を選ぶ
    - 重さが100を超えたらreturnするような再帰関数
    - bit全探索でも良い。
- 貪欲法だと・・・？
  - 大きい順に足していく
    - 62 + 37
  - この答えは間違っている！
    - 50 + 37 + 13の方が大きい！

貪欲法を使える状況

AtCoder

- 探索をするまでもなく、最も良い選択肢を選ぶことで、最良の結果が得られる、と解っていれば、貪欲法を採用して良い。
- 逆に、解っていないときは、迂闊に貪欲法を使ってはいけない！
  - 使う時は出来るだけ慎重に
  - 貪欲法で解ける問題でも、正しくない選び方をしてしまうと、間違った答えを出してしまうことも。

貪欲法　実践

AtCoder

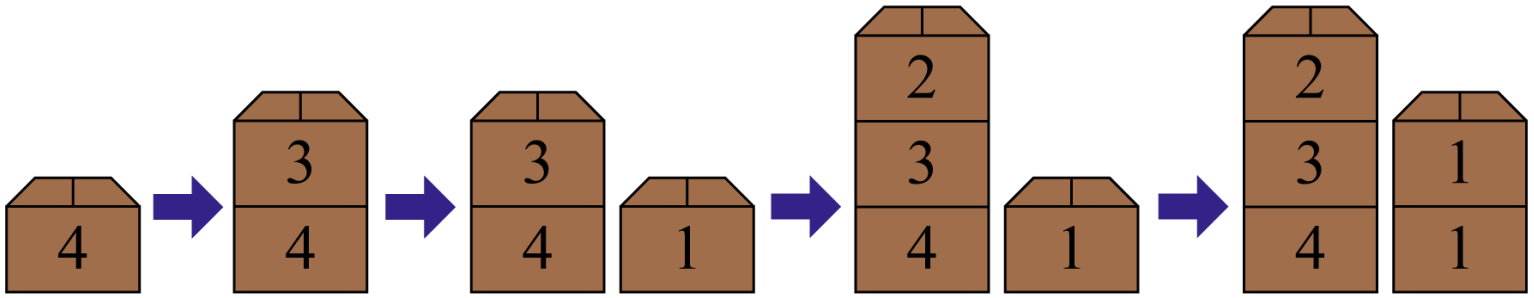
- ARC006 C 積み重ね
  - http://arc006.contest.atcoder.jp/tasks/arc006_3

- 問題概要
  - N個の段ボールを与えられた順番に積み重ねていく
    - N<=50
  - 段ボールは重さを持ち、自分より重い段ボールは積み重ねることが出来ない。
  - 積み重ねられないときは、別の山を作る
  - 段ボールの順番の前後を入れ替えてはならない
  - 段ボールの山を最小にしたい時に、その山の個数を出力せよ

貪欲法　演習

AtCoder

- 問題例
  - N=5 w={4,3,1,2,1}の時
  - 以下のような順序で、2つの山にすることが出来る



貪欲法　演習

AtCoder

- 以下のルールを守って、段ボールを積み重ねていけば良い。
  - 段ボールを積み重ねることが可能な時は、それを積み重ねる。
  - 積み重ねる時は、出来るだけ軽い段ボールの上に重ねる
- この2つを守れば、必ず最小値となる。

  - 証明
    - 次の状態に、出来るだけ重い段ボールが残っていた方が良いので、出来るだけ軽い段ボールに乗せるべき
    - 「新しい山を載せる」は「重さ∞の山に積み重ねる」と投下なので、どんな状況でも最も不利な選択となる。

貪欲法　演習

AtCoder

- 実装方法
  - 軽い段ボールの山から順番に、積み重ねることが可能かどうか調べる
  - 積み重ねることが可能であれば、新しい段ボールの重さをその山に上書きする
  - 不可能であれば、新しい山を作る

貪欲法　実践

AtCoder

- ソースコード
  - http://arc006.contest.atcoder.jp/submissions/153213
  - 入力と初期化

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int n = cin.nextInt();
    int[] w = new int[n];
    for(int i=0;i<n;i++) w[i] = cin.nextInt();

    //乗せる山を作る、初期の重さは∞と十分に見做せるもの
    //最大でもn個の山しか出来ないので、山の数はnで十分
    int[] box = new int[n];
    int INF = 999999;
    for(int i=0;i<n;i++){
        box[i] = INF;
    }
```

貪欲法　実践

AtCoder

- ソースコード
  - http://arc006.contest.atcoder.jp/submissions/153213
  - 貪欲に段ボールを重ねていく

```
    int ret = 0;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            //もし乗せれたら上書きを行う
            if(box[j] >= w[i]){
                box[j] = w[i];
                //ついでに回答の更新も行う。
                ret = Math.max(ret, j + 1);
                break;
            }
        }
    }
    System.out.println(ret);
}
```

貪欲法　演習

AtCoder

- ABC003 AtCoderプログラミング講座
  - http://abc003.contest.atcoder.jp/tasks/abc003_3

- 問題概要
  - N(N≦100)個のビデオがある。‘
  - 高橋君の初期のレーティングは0である。
  - 高橋君のレーティングがCとして、レーティングRのプログラマのビデオを見ると、レーティングが(C+R)/2となる。
  - K(K≦N)個のビデオを見ることが可能
  - 高橋君がビデオを見て到達できる最高のレーティングを求めなさい。

貪欲法　演習

AtCoder

- 解き方
  - 高橋君が、どういう順番でビデオを見るべきかを考えよう！
  - 色々なケースを試してみるのが大切！
  - 順番さえ解ってしまえば後は簡単！

貪欲法 演習

AtCoder

- 出来た人は以下の問題にも挑戦してみよう！
  - ABC005 C おいしいたこ焼きの売り方（普）
    - http://abc005.contest.atcoder.jp/tasks/abc005_3
  - ARC014 C 魂の還る場所（難）
    - http://arc014.contest.atcoder.jp/tasks/arc014_3
  - ARC018 C 席替え（難）
    - http://arc018.contest.atcoder.jp/tasks/arc018_3
  - ARC007 D 破れた宿題（鬼）
    - http://arc007.contest.atcoder.jp/tasks/arc007_4

貪欲法　演習

AtCoder

- 解き方
  - 全通り試すと、N<=100なので、最大100!通り
    - これは到底間に合わない。
  - 組み合わせが膨大な時どうするか？
  - 下記のどちらかを行えば、上手く行くことが多い！
    - 計算を上手くまとめてあげる
      - 動的計画法やメモ化再帰など
      - 今回はこれでは間に合わない
    - 規則性を見つけて調べる必要のあるパターンを減らす
      - 貪欲法や、探索の枝刈など。
      - 今回は貪欲法で可能な方法を考えよう！

貪欲法　演習

AtCoder

- 規則性を見つける
  - どうやって？
  - 規則性を見つけたい時は、色々なケースを手で試してみよう！

貪欲法　演習

AtCoder

- 動画が{1,2,3}の3つ、このうち2つを見る。
  - 1,2と見る→ 1.25
  - 1,3と見る→ 1.75
  - 2,1と見る→ 1
  - 2,3と見る→ 2
    - これが最大
  - 3,1と見る→1.25
    - 3だけ見たほうが大きい
  - 3,2と見る→1.75

貪欲法　演習

AtCoder

- 先ほどのケースを観察して解りそうなこと
  - 大きい方からK個採用すると良い。
    - これは直感的にも自明？
    - もし小さい数を採用していたら、使っていない中でレートの大きいものと取り換えれば、最終的な値は増えそう
  - 見る動画のセットを決めたら、レートの低い動画から順番に見ていくと良い
    - これは本当？？偶然こうなっただけ？？

貪欲法　演習

AtCoder

- {1,2,3}の順番を変えてみる
  - 1,2,3 → 2.125 これが一番大きい！
  - 1,3,2 → 1.875
  - 2,1,3 → 2
  - 2,3,1 → 1.5
  - 3,1,2 → 1.625
  - 3,2,1 → 1.375

貪欲法　演習

AtCoder

- 参考:計算詳細
  - 1 2 3
  - C = 0.0のとき、新たなCはC = (0.0+1)/2 = 0.5
  - C = 0.5のとき、新たなCはC = (0.5+2)/2 = 1.25
  - C = 1.25のとき、新たなCはC = (1.25+3)/2 = 2.125
  - 1 3 2
  - C = 0.0のとき、新たなCはC = (0.0+1)/2 = 0.5
  - C = 0.5のとき、新たなCはC = (0.5+3)/2 = 1.75
  - C = 1.75のとき、新たなCはC = (1.75+2)/2 = 1.875
  - 2 1 3
  - C = 0.0のとき、新たなCはC = (0.0+2)/2 = 1.0
  - C = 1.0のとき、新たなCはC = (1.0+1)/2 = 1.0
  - C = 1.0のとき、新たなCはC = (1.0+3)/2 = 2.0
  - 2 3 1
  - C = 0.0のとき、新たなCはC = (0.0+2)/2 = 1.0
  - C = 1.0のとき、新たなCはC = (1.0+3)/2 = 2.0
  - C = 2.0のとき、新たなCはC = (2.0+1)/2 = 1.5
  - 3 1 2
  - C = 0.0のとき、新たなCはC = (0.0+3)/2 = 1.5
  - C = 1.5のとき、新たなCはC = (1.5+1)/2 = 1.25
  - C = 1.25のとき、新たなCはC = (1.25+2)/2 = 1.625
  - 3 2 1
  - C = 0.0のとき、新たなCはC = (0.0+3)/2 = 1.5
  - C = 1.5のとき、新たなCはC = (1.5+2)/2 = 1.75
  - C = 1.75のとき、新たなCはC = (1.75+1)/2 = 1.375

貪欲法　演習

AtCoder

- 考察：なぜ小さい方から順番に見ていくべきなのか？
  - 最初に見る動画のレートをa、次に見る動画をb、最後に見る動画をcとする
  - 高橋君の初期レートは0
  - １つ動画を見た後のレートはa/2
  - ２つ動画を見た後のレートはb/2 + a/4
  - ３つ動画を見た後のレートはc/2 + b/4 + a/8
- 後から見た動画の方が影響力が強くなる。

貪欲法　演習

AtCoder

- 以上より、行うべき手順は以下のようになる
  - ビデオをレーティング順にソートする
  - 大きい方からK個の数字を取り出し、小さい方から順に1つずつシミュレーションしてあげる
    - 小さい方から取り出しているので、途中で高橋君のレートより動画のレートが小さくなることはありえないため、必ずK個の動画を閲覧して良い。

- ソースコード
  - http://abc003.contest.atcoder.jp/submissions/153212

```
void run()
{
    Scanner cin = new Scanner(System.in);
    int N = cin.nextInt();
    int K = cin.nextInt();
    int[] r = new int[N];
    for(int i=0; i<N; i++){
        r[i] = cin.nextInt();
    }

    double ret = 0;
    //上からK個を、小さい順に使う
    Arrays.sort(r);
    for(int i=N-K;i<N;i++){
        ret = (ret + r[i]) / 2;
    }
    System.out.println(ret);
}
```

本日のまとめ

本日のまとめ

AtCoder

- BitDP
  - 集合を扱う動的計画法が書けるようになった！
- 貪欲法
  - 探索するまでもない問題を解けるようになった！
  - 貪欲法を使って良い場面、悪い場面がある程度見極められるようになった！