2024 年情報オリンピック日本委員会主催 レギオ講習会 静岡会場 初中級編（2 日目）講義資料

[オリエンテーション]

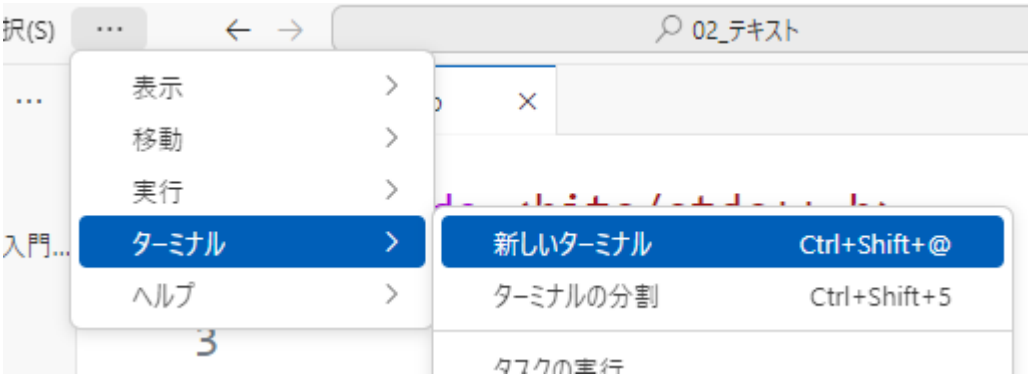
本講習会では，C++ が対象言語です．
自身で C++ のコンパイル・実行環境がない方は
(1) https://wandbox.org/ を利用する
　下記の様に，ブラウザ上で C++の編集，コンパイル，実行ができます．

(2)https://strawberryperl.com/ から，Strawberry Perl をインストールする．これで，gcc，g++ という，C 言語および C++のプログラムをコンパイルできる環境も同時にインストールされる
　（自宅等の PC の場合 VS Code 等のエディタ上で，端末を起動すればローカル環境で演習可能です

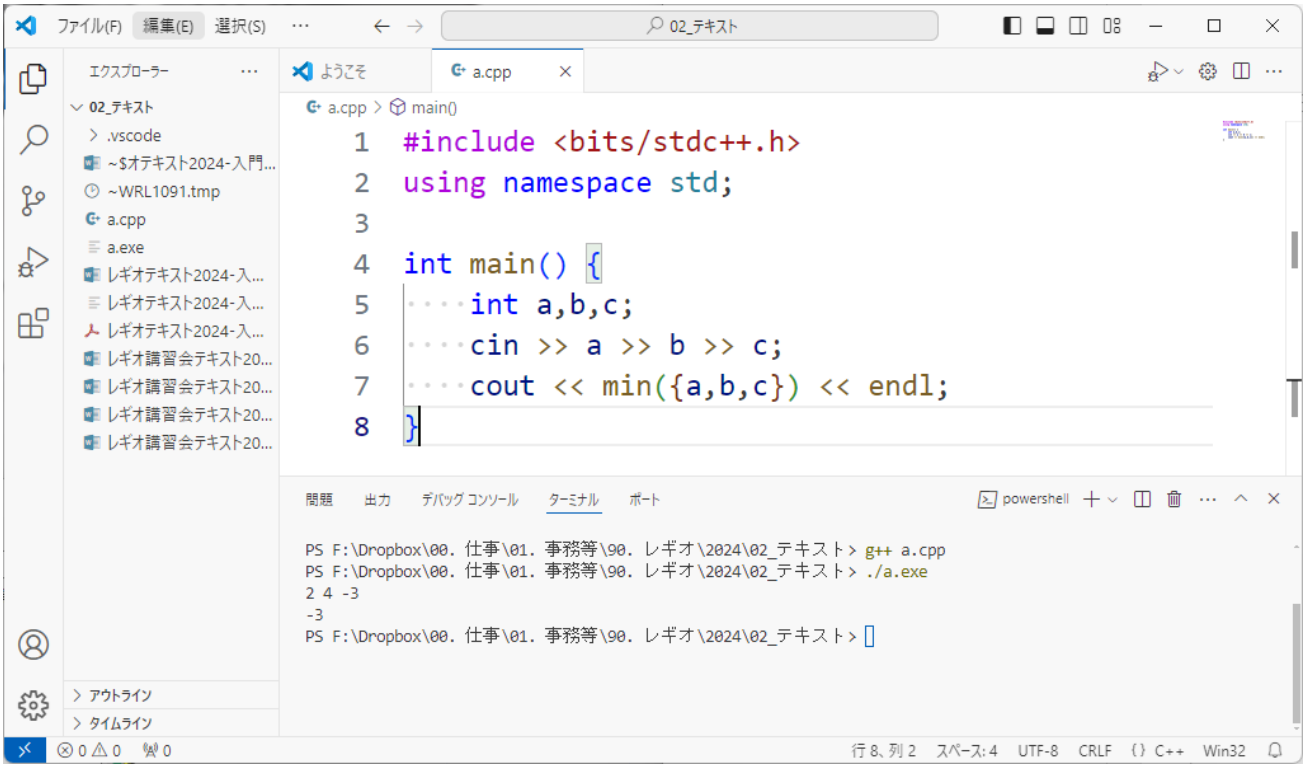
```
// template (g++ の場合．clang を使う場合は下記はそのままだと NG)
#include <bits/stdc++.h>
using namespace std;

int main() {
   cout << “Hello worlds!” << endl;
}
```

Strawberry Perl インストール後，Windows で VSCode で演習をする場合の例
下記は，プログラムを作成後，メニューから



新しいターミナルを選んで，



ターミナルの方で，
> g++ 作ったファイル名
でコンパイル後，

> ./a.exe
で実行できる．
入力は実際にキーボードから数字等を入力すれば良い

① 10:00〜10:50『JOI2022/2023 一次予選にバーチャル参加しよう』

https://xskogure.github.io/regio2024shizuoka/second.html
の①を参照のこと

JOI2022/2023 一次予選にバーチャル参加してみましょう.
https://atcoder.jp/contests/joi2023yo1a を開き『バーチャル参加』をクリックしたあと，下記の 4 つの問題を 50 分でできる限り解いてみてください(本番は 80 分！).

② 11:00～12:00『繰り返しの方法を確認しよう』＆『リストの使い方を確認しよう』（1 日目の復習）

https://xskogure.github.io/regio2024shizuoka/second.html
の②を参照のこと

例題 ■ Rightmost (AtCoder Beginner Contest 276 問題 A)
英小文字からなる文字列 S が与えられます。
S に a が現れるならば最後に現れるのが何文字目かを出力し、現れないならば −1 を出力してください。
ポイント：最後に現れる→ 末尾から探索すると楽！

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

int main() {
    int i, len, ap=-1;
    string S;
    cin >> S;
    len = S.size();
    for(i=len-1;i>=0;i--) {
        if (S.at(i) == 'a') {
            ap = i+1;
            break;
        }
    }
    cout << ap << endl;
}
```

例題 ■ IOI 文字列 (JOI 2020/2021 一次予選 (第 3 回) 問題 B)

IOI 文字列とは次の条件をすべて満たす文字列である.

・長さが奇数である.

・各文字は I または O で，これらが交互に連なる.

・1 文字目は I である.

あなたは次の操作を 0 回以上繰り返すことができる.

・文字列 S の文字を 1 つ選び，好きな英大文字に変更する.

文字列 S を IOI 文字列にするのに必要な操作の回数の最小値を求めよ.

ポイント：奇数番目に'I'が，偶数番目に'O' が「ない」数を数えればいい

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

int main() {
    int N,i,count=0;
    string S;
    cin >> N >> S;
    for(i=0;i<N;i++) {
        if      (i%2==0 && S.at(i) != 'I') count++;
        else if (i%2==1 && S.at(i) != 'O') count++;
    }
    cout << count << endl;
}
```

例題 ■ 分割 (JOI 2020/2021 一次予選 (第 2 回) 問題 C)

最大値で数列を分割したとき，最大値より前にある値の和と，最大値より後ろにある値の和を出力せよ．

ポイント：最大値の「場所」を調べて，その前までと，その後からの和を別途求めれば良い．

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

int main() {
    int N,sumBefore=0,sumAfter=0,maxPos,i;
    cin >> N;
    vector<int> A(N);
    cin >> A.at(0);
    maxPos=0;
    for(i=1;i<N;i++) {
        cin >> A.at(i);
        if (A.at(maxPos) < A.at(i)) {
            maxPos = i;
        }
    }
    for(i=0;i<=maxPos-1;i++) sumBefore+=A.at(i);
    for(i=maxPos+1;i<N;i++)  sumAfter +=A.at(i);

    cout << sumBefore << endl;
    cout << sumAfter << endl;
}
```

例題 ■ 最頻値 (JOI 2019/2020 一次予選 (第 2 回) 問題 C)

問題文

長さ N の数列 $A_1, A_2, \ldots, A_N$ が与えられる．この数列の各項は 1 以上 M 以下の整数である．

長さ M の新たな数列 $B_1, B_2, \ldots, B_M$ を以下のように定義する．

- 各 j $(1 \leqq j \leqq M)$ に対して，B_j の値は $A_i = j$ を満たす整数 i $(1 \leqq i \leqq N)$ の個数に等しい．

$B_1, B_2, \ldots, B_M$ の最大値を求めよ．

ポイント：難しく書いてあるが最頻出する数を求めれば良い．さらに，数字の範囲が 1<=Ai<=M (最大 100) なので，101 要素の配列があれば良い．

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

int main() {
    int N, M, i, num, max;
    cin >> N >> M;
    vector<int> B(M+1,0);
    for(i=0;i<N;i++) {
        cin >> num;
        B.at(num) = B.at(num) + 1;
    }
    max = B.at(1);
    for(i=2;i<=M;i++) {
        if (max < B.at(i)) max = B.at(i);
    }
    cout << max << endl;
}
```

それでは演習！

https://xskogure.github.io/regio2024shizuoka/second.html
の②を参照のこと

③ 13:00～13:50『辞書順・整列』を使いこなそう！『待ち行列とスタック』を理解しよう！

C++ 文法の確認

[整列]

```
vector<int> A;

sort(A.begin(), A.end());  // 小さい順(昇順)
sort(A.rbegin(), A.rend()); // 大きい順(降順)
```

[辞書順]

“A” と “AB” では “A” のほうが先
“AB” と “AA” では “AA” のほうが先
“A” < “AB” のように比較できる！

[待ち行列]

```
queue<int> q;
q.push(登録); // データを末尾に登録
q.front(); // 先頭のデータにアクセス
q.pop(); // 先頭のデータを削除
q.empty(); // 待ち行列が空であれば true
```

[スタック]

```
stack<int> s;
s.push(登録); // データを top に登録
s.top(); // top のデータにアクセス
s.pop(); // top のデータを削除
s.empty();  // スタックが空なら true
```

[deque] (待ち行列とスタックの両方の使い方が可能

```
deque<int> dq;
dq.push_back(登録); // データを末尾に登録
dq.push_front(登録); // データを先頭に登録
dq.front(); // 先頭のデータにアクセス
dq.back();  // 末尾のデータにアクセス
dq.pop_back(); // 末尾のデータを削除
dq.pop_front(); // 先頭のデータを削除
```

こちらも確認！

- O - 1.14.STL の関数 (https://atcoder.jp/contests/apg4b/tasks/APG4b_o)
    - Sort の説明等
- AA - 3.03.STL のコンテナ (https://atcoder.jp/contests/apg4b/tasks/APG4b_aa)
    - queue, stack, deque の説明等
- AI - 4.04.イテレータ (https://atcoder.jp/contests/apg4b/tasks/APG4b_ai)

https://xskogure.github.io/regio2024shizuoka/second.html
の③を参照のこと

例題 ■ 辞書式順序 (AtCoder Beginner Contest 007 問題 B)

問題文

文字列 A が与えられる。小文字アルファベット(a-z)のみを使って辞書順比較したとき文字列 A より小さいものを1つ何でも良いので出力せよ。ただし、文字列は 1 文字以上 100 文字以下でなければならない。もし存在しない場合は "-1" を出力せよ。

- それぞれの文字列の同じ位置同士を先頭から比較していって、初めて不一致になったら、その文字同士の(アルファベットでの)比較結果が文字列の全体の比較結果である。例えば、”$abcd$” と”ax” を比較すると、2 文字目で、$'b' <' x'$ となるので、”$abcd$” $<$ ”ax” である。
- もし、比較している途中で片方の文字列が尽きてしまったら、文字列の長さが短い方が小さい。例えば”ab” $<$ ”abc” である。

ポイント：C++ では辞書順は，比較演算子で調べられる

・文字数が 2 文字以上なら，末尾の文字を削ったものを作る

・文字数が 1 文字であれば，ascii コードを 1 減らして，一文字前のアルファベットにする

→ただし，’a’ より前の文字は存在しないので -1 を出力する

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

int main() {
    string A;
    cin >> A;
    int len = A.size();
    if (len > 1) {
        // 先頭から len-1 文字分出力（末尾を削る）
        cout << A.substr(0,len-1) << endl;
    } else { // len==1 のはず
        char ch = A.at(0);
        if (ch == 'a') { // 'a' より辞書順が前のものは作れない
            cout << "-1" << endl;
        } else {
            cout << (char)(ch-1) << endl;
        }
    }
}
```

別解：こちらのほうがより簡単

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

int main() {
    string A;
    cin >> A;
    if (A == “a”) cout << “-1” << endl;
    else cout << “a” << endl;
}
```

例題 ■ 心配性な富豪、ファミリーレストランに行く。(AtCoder Beginner Contest 009 問題 B)

問題文

とはいえ、せっかくだから高いものを選んでその味をみてみたいというのも確かである。そうだ、そういうことなら、この店で 2 番目に高い料理を注文することにしよう。そう思って料理の金額を書き出してみたが、料理の種類が多いために 2 番目に高いものを探すのはなかなか骨が折れる。自分で探すかわりに、プログラムを書いてなんとかできないだろうか？

おっと、プログラムを書き始める前にひとつ言っておくが、最も高い金額の料理が複数あるときには注意してもらいたい。というのは、たとえば 4 種類の料理があり、それぞれの金額が 100 円、200 円、300 円、300 円であったときには、2 番目に高いものというのは 200 円の料理になるということだ。

ポイント：ただ整列をして 2 番目の値を出力してしまうと，同じ料金があったときに間違えてしまう．そこで，大きい順に整列したあと，一番大きい値と違う値を発見したらその値を出力する（小さい順に整列したあと，後ろから調べても OK）

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

int main() {
    int N;
    cin >> N;
    vector<int> A(N);
    for(int i=0; i<N; i++) cin >> A.at(i);
    sort(A.rbegin(), A.rend()); // 大きい順に整列
    int i;
    for (i=1; i<N; i++) {
        // 最初に一個前と違う値段が発見されればそれが 2 番目のやつ
        if (A.at(i-1) != A.at(i)) {
            cout << A.at(i) << endl;
            break;
        }
    }
}
```

例題 ■ 3 人でカードゲームイージー (AtCoder Beginner Contest 045 問題 B)

問題文

A さん、B さん、C さんの 3 人が以下のようなカードゲームをプレイしています。

- 最初、3 人はそれぞれ a、b、c いずれかの文字が書かれたカードを、何枚か持っている。これらは入力で与えられた順番に持っており、途中で並べ替えたりしない。
- A さんのターンから始まる。
- 現在自分のターンである人がカードを 1 枚以上持っているならば、そのうち先頭のカードを捨てる。その後、捨てられたカードに書かれているアルファベットと同じ名前の人 (例えば、カードに a と書かれていたならば A さん) のターンとなる。
- 現在自分のターンである人がカードを 1 枚も持っていないならば、その人がゲームの勝者となり、ゲームは終了する。

3 人が最初に持っているカードがそれぞれ先頭から順に与えられます。具体的には、文字列 S_A、S_B、S_C が与えられます。文字列 S_A の i 文字目 ($1 \leq i \leq |S_A|$) に書かれている文字が、A さんの持っている中で先頭から i 番目のカードに 書かれている文字です。文字列 S_B、 S_C についても同様です。

最終的に誰がこのゲームの勝者となるかを求めてください。

ポイント：3 人のカード状況を 3 つの待ち行列で管理すると良い.

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

int main() {
    string SA,SB,SC;
    cin >> SA >> SB >> SC;
    // A,B,C それぞれの待ち行列を作る
    queue<char> QA, QB, QC;
    for(int i=0; i<(int)SA.size(); i++) QA.push(SA.at(i));
    for(int i=0; i<(int)SB.size(); i++) QB.push(SB.at(i));
    for(int i=0; i<(int)SC.size(); i++) QC.push(SC.at(i));

    char pos = 'a';
    while(1) {
        // cout << "pos: " << pos << endl;
        if       (pos == 'a') {
            if (QA.empty()) { cout << "A" << endl; break; }
            else { pos = QA.front(); QA.pop(); }
        } else if (pos == 'b') {
            if (QB.empty()) { cout << "B" << endl; break; }
            else { pos = QB.front(); QB.pop(); }
        } else if (pos == 'c') {
            if (QC.empty()) { cout << "C" << endl; break; }
            else { pos = QC.front(); QC.pop(); }
        }
    }
}
```

問題文

しぐはキーボードを製作しました。シンプルさを極限まで追求したこのキーボードには、`0`キー、`1`キー、バックスペースキーの 3 つしかキーがありません。

手始めに、しぐはこのキーボードで簡単なテキストエディタを操作してみることにしました。このエディタには常に一つの文字列が表示されます（文字列が空のこともあります）。エディタを起動した直後では、文字列は空です。キーボードの各キーを押すと、文字列が次のように変化します。

- `0`キー: 文字列の右端に文字`0`が挿入される。
- `1`キー: 文字列の右端に文字`1`が挿入される。
- バックスペースキー: 文字列が空なら、何も起こらない。そうでなければ、文字列の右端の 1 文字が削除される。

しぐはエディタを起動し、これらのキーを何回か押しました。しぐが押したキーを順番に記録した文字列 s が与えられます。s の中の文字`0`は`0`キー、文字`1`は`1`キー、文字`B`はバックスペースキーを表します。いま、エディタの画面にはどのような文字列が表示されているでしょうか？

ポイント：stack で一旦データを管理すると良い．ただこの方法だと，stack の最終状態を表示する方法がないので，vector に一旦コピーしてから表示している（deque で実装すると at() が使えるので vector にコピーする必要がないかも）

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

int main() {
    string s;
    cin >> s;
    stack<char> ST;
    for(int i=0; i<(int)s.size(); i++) {
        char ch = s.at(i);
        if (ch == 'B') {
            if (!ST.empty()) {
                ST.pop(); // B なら最後の文字を削除
            }
        }
        else ST.push(ch);  // 01 なら文字を最後に追加
    }
    // stack をそのまま表示する方法はないので，vecter にコピー
    vector<char> O;
    while(!ST.empty()) {
        char ch = ST.top(); ST.pop();
        O.push_back(ch);
    }
    // 右から左の順番なので注意
    for(int i=(int)O.size()-1; i>=0;i--){
        cout << O.at(i);
    }
    cout << endl;
}
```

演習問題は
https://xskogure.github.io/regio2024shizuoka/second.html
の③を参照のこと

④ 14:00〜14:50『全探索・動的計画法』に触れよう！

全探索：
(広義) 可能性のあるものをすべて調べること
例えば，考えるべきパラメータが 3 種類ある(それぞれの項目数が L， N， M 個)場合，この 3 つがすべて独立(関係がない) のであれば，L*N*M の組み合わせすべてを調べる手法

動的計画法：
ある地点 n の最適解は，n-k 地点の最適解と，n-k+1〜n の(単体の)コストを足したものが最小になる組み合わせで計算した値で求められる （つまり全探索する必要はなく，先頭から順番に最適解を求めれば良い．1 次元で扱う場合もあれば，2 次元で扱う場合もある．2 次元の場合，どちらを先に計算するかは一般的には自由．

こちらも確認！
E869120 さんの Qiita の記事を読んでおこう！

例題 ■ Palace (AtCoder Beginner Contest 113 問題 B)

問題文

ある国で、宮殿を作ることになりました。

この国では、標高が x メートルの地点での平均気温は $T - x \times 0.006$ 度です。

宮殿を建設する地点の候補は N 個あり、地点 i の標高は H_i メートルです。

joisinoお姫様は、これらの中から平均気温が A 度に最も近い地点を選んで宮殿を建設するようにあなたに命じました。

宮殿を建設すべき地点の番号を出力してください。

ただし、解は一意に定まることが保証されます。

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

int main() {
    int N,T,A;
    cin >> N >> T >> A;
    vector<int> H(N+1);
    for(int i=1; i<=N; i++) cin >> H.at(i);
    int minPos=1;
    double minDiff = A - ((double)T - H.at(1) * 0.006);
    // 単純に見えるがこのようにすべての可能性を全部計算して
    // 求めるものはすべて全探索といえる.
    for(int i=2; i<=N; i++) {
        double diff = A - ((double)T - H.at(i) * 0.006);
        if (abs(minDiff) > abs(diff)) {
            minDiff = diff;
            minPos = i;
        }
    }
    cout << minPos << endl;
}
```

問題文

正整数 X が与えられます。 X 以下の最大のべき乗数を求めてください。ただし、べき乗数とは、ある 1 以上の整数 b と 2 以上の整数 p を使って b^p とかける整数のことを指すこととします。

ポイント：こちらも `b` と `p` の全組み合わせを計算すれば良い．ただし，明らかに `1000` を超える計算はする必要がないので少しだけ無駄を排除したほうがいい．

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

int main() {
    int X;
    cin >> X;
    int max = 1; //  1 は該当数．下の繰り返しで b=1 とすると
    // 無駄な計算が入る(1^2, 1^3, 1^4 等すべて 1) ので省略

    // このように可能性をすべて計算してそこから解を探す
    // ものを広義で全探索と呼ぶ
    for (int b=2; b<=sqrt(X); b++) {
        for (int p=2; p<=9; p++) { // 2^10=1024
            int n = (int)pow(b,p);
            if (n>X) break;
            if (max<n) max=n;
            // cout << b << "^" << p << "=" << n << endl;
        }
    }
    cout << max << endl;
}
```

問題文

N 個の足場があります。足場には $1, 2, \ldots, N$ と番号が振られています。各 $i\,(1 \leq i \leq N)$ について、足場 i の高さは h_i です。

最初、足場 1 にカエルがいます。カエルは次の行動を何回か繰り返し、足場 N まで辿り着こうとしています。

- 足場 i にいるとき、足場 $i+1$ または $i+2$ へジャンプする。このとき、ジャンプ先の足場を j とすると、コスト $|h_i - h_j|$ を支払う。

カエルが足場 N に辿り着くまでに支払うコストの総和の最小値を求めてください。

ポイント：2 個前からのジャンプと，1 個前からのジャンプのどちらか良い方を選ぶ，，を動的計画法で求めれば良い．

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

int main() {
    int N;
    cin >> N;
    vector<int> h(N+1);
    for(int i=1;i<=N;i++) cin >> h.at(i);
    vector<int> cost(N+1,0);

    //動的計画法(DP)を使う
    // DP を使うことで，最適解が，　（一回前までのジャンプの最適解
    // + 今回のジャンプ）の可能性のうち，一番コストが小さいもの
    // で求まる，というもの

    // まず，1 はその場（ジャンプなし）でコスト 0
    // 2 も選択肢なし(1 からのジャンプしかない)
    cost.at(1) = 0;
    cost.at(2) = abs(h.at(2)-h.at(1));
    for (int i=3; i<=N; i++) {
        // 2 個前までの累積コスト＋2 個前からここまでのジャンプのコスト
        int c2 = cost.at(i-2) + abs(h.at(i)-h.at(i-2));
        // 1 個前までの累積コスト＋1 個前からここまでのジャンプのコスト
        int c1 = cost.at(i-1) + abs(h.at(i)-h.at(i-1));
        if (c1<c2) cost.at(i) = c1; // 小さい方を採用
        else       cost.at(i) = c2;
    }
    cout << cost.at(N) << endl;
}
```

演習問題は

2024 年情報オリンピック日本委員会主催 レギオ講習会 静岡会場 初中級編（2 日目）講義資料

https://xskogure.github.io/regio2024shizuoka/second.html
の④を参照のこと

⑤ 15:00〜16:10『全探索・深さ優先探索・幅優先探索』に触れよう！

深さ優先探索：例えば，4 方向探索をする際にある方向にひたすら先に進み行き止まりになったら一つ前に戻って次の方向を探すような手法
再帰呼び出し，スタックを利用した方法が考えられる

幅優先探索：例えば，4 方向探索をする際に，4 方向すべて並列に 1 個だけ進んだ状態を作りそこからさらに 4 方向進む例を順番に調べる手法
待ち行列を利用した方法が考えられる

深さ優先探索・幅優先探索については，E869120 さんの Qiita の記事を読んでおこう！
深さ優先探索については，例題に取り上げた問題にも解説が乗っているのでそちらを見よう．

例題 ■ 深さ優先探索 (Coder Typical Contest 001 問題 A)

問題文

この問題は、講座用問題です。ページ下部に解説が掲載されています。

高橋君の住む街は長方形の形をしており、格子状の区画に区切られています。 長方形の各辺は東西及び南北に並行です。 各区画は道または塀のどちらかであり、高橋君は道を東西南北に移動できますが斜めには移動できません。 また、塀の区画は通ることができません。

高橋君が、塀を壊したりすることなく道を通って魚屋にたどり着けるかどうか判定してください。

方法 1 ：再帰呼び出しを使う方法.

座標を管理したいので，pair を利用している.

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

#define rep(i, s, n) for (int i = (s); i < (int)(n); i++)

int H,W;
int sx,sy,gx,gy;// スタート位置とゴール位置
// 迷路情報保存
vector<vector<char>> c(501,vector<char>(501));
// 到達できたか
vector<vector<bool>> isReached(501,vector<bool>(501,false));

void search(int cx, int cy) {
    if( isReached.at(gy).at(gx)) return; // g 到達済みならもう探す必要なし
    if (cx < 1 || cx > W || cy < 1 || cy > H) return;
    char ch = c.at(cy).at(cx);
    // cout << "(" << cx << "," << cy << ")= " << ch << endl;
    if (ch == '#') return; // 移動不可
    if (isReached.at(cy).at(cx)) return;// すでに到達済み
    isReached.at(cy).at(cx) = true; // 到達済みフラグ
    search(cx-1,cy); // 深さ優先探索の基本形は再帰呼び出し
    search(cx+1,cy); // ある方向に行けるだけ行き，いけなくなったら
    search(cx,cy-1); // 次の方向に（この場合はまず左側）
    search(cx,cy+1); // 左 右 上 下 の順番
}


int main() {
    cin >> H >> W;
    rep(y,1,H+1) rep(x,1,W+1) {
        cin >> c.at(y).at(x);
        if      (c.at(y).at(x) == 's') { sx = x; sy = y; }
        else if (c.at(y).at(x) == 'g') { gx = x; gy = y; }
    }
    search(sx, sy); // スタート地点から始める
    if (isReached.at(gy).at(gx)) { //ゴール位置に到達できたら
        cout << "Yes" << endl;
    } else {
        cout << "No" << endl;
    }
}
```

方法２：スタックを使う方法

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

#define rep(i, s, n) for (int i = (s); i < (int)(n); i++)

int H,W;
int sx,sy,gx,gy;// スタート位置とゴール位置
// 迷路情報保存
vector<vector<char>> c(501,vector<char>(501));
// 到達できたか
vector<vector<bool>> isReached(501,vector<bool>(501,false));

// 深さ優先探索を再帰で行わない場合は stack を使うのが良い
stack<pair<int,int>> ps;

void search(int cx, int cy) {
    ps.push(make_pair(cx,cy));
    while( !ps.empty()) { // スタックを使うことで再帰呼び出しなし
        pair<int,int> pos = ps.top(); ps.pop();
        int cx = pos.first;
        int cy = pos.second;
        if( isReached.at(gy).at(gx)) break; // g 到達済みならもう探す必要なし
        if (cx < 1 || cx > W || cy < 1 || cy > H) continue;
        char ch = c.at(cy).at(cx);
        // cout << "(" << cx << "," << cy << ")= " << ch << endl;
        if (ch == '#') continue; // 移動不可
        if (isReached.at(cy).at(cx)) continue;// すでに到達済み
        isReached.at(cy).at(cx) = true; // 到達済みフラグ
        ps.push(make_pair(cx-1,cy)); // ４方向をスタックに登録
        ps.push(make_pair(cx+1,cy)); // ４方向をスタックに登録
        ps.push(make_pair(cx,cy-1)); // ４方向をスタックに登録
        ps.push(make_pair(cx,cy+1)); // ４方向をスタックに登録
    }
}


int main() {
    cin >> H >> W;
    rep(y,1,H+1) rep(x,1,W+1) {
        cin >> c.at(y).at(x);
        if      (c.at(y).at(x) == 's') { sx = x; sy = y; }
        else if (c.at(y).at(x) == 'g') { gx = x; gy = y; }
    }
    search(sx, sy); // スタート地点から始める
    if (isReached.at(gy).at(gx)) { //ゴール位置に到達できたら
        cout << "Yes" << endl;
    } else {
        cout << "No" << endl;
    }
}
```

例題 ■ 幅優先探索 (Coder Typical Contest 002 問題 A)

問題文

たかはし君は迷路が好きです。今、上下左右に移動できる二次元盤面上の迷路を解こうとしています。盤面は以下のような形式で与えられます。

- まず、盤面のサイズと、迷路のスタート地点とゴール地点の座標が与えられる。
- 次に、それぞれのマスが通行可能な空きマス(.)か通行不可能な壁マス(#)かという情報を持った盤面が与えられる。盤面は壁マスで囲まれている。スタート地点とゴール地点は必ず空きマスであり、スタート地点からゴール地点へは、空きマスを辿って必ずたどり着ける。具体的には、入出力例を参考にすると良い。

今、彼は上記の迷路を解くのに必要な最小移動手数を求めたいと思っています。どうやって求めるかを調べていたところ、「幅優先探索」という手法が効率的であることを知りました。幅優先探索というのは以下の手法です。

- スタート地点から近い(たどり着くための最短手数が少ない)マスから順番に、たどり着く手数を以下のように確定していく。説明の例として図1の迷路を利用する。

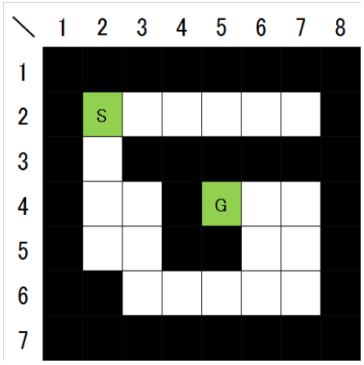


図1. 説明に用いる盤面

ポイント：幅優先探索では，待ち行列（キュー）を利用する
座標＋コストを保持したいので，tuple (3 つ組) を利用している．

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

#define rep(i, s, n) for (int i = (s); i < (int)(n); i++)

int R,C;
int sx,sy,gx,gy;// スタート位置とゴール位置
// 迷路情報保存
vector<vector<char>> c(51,vector<char>(51));
// S からの最短距離
vector<vector<int>> cost(51,vector<int>(51,2501));

// 幅優先探索では待ち行列 queue を使うのが良い
queue<tuple<int,int,int>> pq;

void search() {
    pq.push(make_tuple(sx,sy,0));
    while( !pq.empty()) { // 待ち行列が空になるまで繰り返す
        tuple<int,int,int> pos = pq.front(); // 先頭のデータを取り出す
        pq.pop(); // 先頭からデータを削除
        int cx = get<0>(pos);
        int cy = get<1>(pos);
        int cc = get<2>(pos);
        if (cx < 1 || cx > C || cy < 1 || cy > R) continue;
        char ch = c.at(cy).at(cx);
        //cout << "(" << cx << "," << cy << "," << cc << ")=" << ch << endl;
        if (ch == '#') continue; // 移動不可
        // 幅優先探索のお陰で初回の計算が最短であることが保証される
        if (cost.at(cy).at(cx)==2501) cost.at(cy).at(cx) = cc;
        else continue; // 2501 じゃないということはすでに到達済み
        cc = cost.at(cy).at(cx);
        pq.push(make_tuple(cx-1,cy,cc+1)); // ４方向を待ち行列に登録
        pq.push(make_tuple(cx+1,cy,cc+1)); // 隣への移動なので
        pq.push(make_tuple(cx,cy-1,cc+1)); // コスト＋１を保存
        pq.push(make_tuple(cx,cy+1,cc+1)); //
    }
}


int main() {
    cin >> R >> C;
    cin >> sy >> sx;
    cin >> gy >> gx;
    rep(y,1,R+1) rep(x,1,C+1) {
        cin >> c.at(y).at(x);
    }
    cost.at(sy).at(sx);
    search(); // スタート地点(cost 0)から始める
    cout << cost.at(gy).at(gx) << endl;
}
```

まず，JOI 2023/2024 二次予選 過去問の A,B にチャレンジ．

演習問題は
https://xskogure.github.io/regio2024shizuoka/second.html
の④を参照のこと