YouTubeのプレイリスト再生サイト

概要

YouTubeのプレイリストを再生するウェブサイトです。
拡張機能やpc版サイトを利用するために、ブラウザでYouTubeを開く事があると思います。
YouTubeのプレイリスト機能は動画数が増えるとシャッフル機能がまともに働かなくなるので、ブラウザ版YouTubeの代わりとして、このようなサイトを作成しました。

（作った後に気づきましたが、ブラウザ版YouTube Musicではまともにシャッフルできたので、存在意義のないサイトになっています）

使用言語・技術

HTML・CSS
JavaScript
YouTube Data API v3
YouTube Iframe API

機能

再生リスト取得・再生
並び替え
シャッフル
範囲検索
指定時間の長さのプレイリスト作成

使用したアルゴリズム

バブルソート (動画並び替え)

動画数は多くても1000程度を想定しています。
はじめに作ったバブルソートで十分高速に動いたので、そのままバブルソートを採用しました。
(最終的に1クリックで2回ソートするコードになってしまったため、動作速度が遅くなってしまいました)

動的計画法 (プレイリスト作成)

入力された時間に最も近くなる動画時間の組み合わせを探索しています。
(例10分が入力されたら、5分+2分+3分みたいな組み合わせを探索する。)

指定された時間より少し上の範囲までの動画組み合わせによる総時間を全マッピングして、そこからベスト値を探しています。
ベストな値が複数あった場合は、その中からランダムに選んでプレイリストにしています。

学んだこと・感じたこと

現代のコンピュータは早いなと感じました。
今回のコードではバブルソートと、枝切りしない動的計画法を用いていますが、どちらも高速に動作しました。
実務では枝切り動的計画法や、クイックソートなどは、凄く大きいものや高頻度で何度も並び替えるような場面がないとあまり恩恵を感じられないのではないか、と思いました。

ソースコード

index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>YouTube ランダム再生</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/modern-css-reset/dist/reset.min.css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  <link rel="stylesheet" href="style.css">
</head>
```

```
<body class="text-center d-flex flex-column align-items-center m-auto gap-3 px-3">
  <h1 class="fs-2 fw-bold">YouTube プレイリスト再生</h1>

  <h2 class="fs-4">プレイリストID検索</h2>
  <details class="accordion w-75">
    <summary>Id取得法</summary>
    <div class="content fs-6 d-flex flex-column" >
      <span class="align-self-start">YouTubeで使いたい再生リストを開いて</span>
      <span class="align-self-center">https://www.youtube.com/playlist?list=xxxx</span>
      <span class="align-self-end">のxxxx部分をここに貼り付けてください</span>
    </div>
  </details>
  <form id="playlistIdForm" class="d-flex align-items-center justify-content-center gap-2 w-100">
    <input id="playlistId" class="form-control form-control-lg w-75" name="playlistId" placeholder="プレイリストID"/>
    <button class="btn btn-lg btn-outline-primary fw-bold" type="submit">検索</button>
  </form>
  <div id="idError" class="error" hidden>IDが不正です</div>

  <h2 class="fs-4">絞り込み</h2>
  <form id="searchForm" class="d-flex align-items-center justify-content-center">
    <input id="minViews" class="form-control w-25" type="number" name="minViews" min="0" placeholder="再生回数"/>
    <span class="fs-4 mx-1">~</span>
    <input id="maxViews" class="form-control w-25" type="number" name="maxViews" min="0"/>
    <span class="fs-6 ms-1 me-2">万回</span>
    <button class="btn btn-outline-primary fw-bold" type="submit">絞り込み</button>
  </form>

  <h2 class="fs-4">n分のプレイリスト作成</h2>
  <form id="createPlaylistForm">
    <div class="d-flex align-items-center">
      <input id="createPlaylistTime" class="form-control w-50" type="number" name="createPlaylistTime" min="0"
placeholder="プレイリストの長さ"/>
      <span class="fs-6 ms-1 me-2">分</span>
      <button class="btn btn-outline-primary fw-bold" type="submit">作成する</button>
    </div>
    <div id="playlistTotalTime"></div>
  </form>

  <div class="d-flex gap-3">
    <select class="form-select w-auto border-dark fw-bold" name="sort" id="sort" >
      <option value="default">並び替え(デフォルト)</option>
      <option value="viewCountAsc">再生回数(少ない順)</option>
      <option value="viewCountDesc">再生回数(多い順)</option>
    </select>
    <button class="btn btn-outline-primary fw-bold w-auto" id="shuffle">シャッフル</button>
    <button class="btn btn-outline-dark fw-bold w-auto" id="reset" type="button">全てクリア</button>
  </div>

  <div class="d-flex flex-column align-items-center" style="position: relative; width: 40rem; max-width: 100%;">
    <div style="width: 100%; aspect-ratio: 16/9; ">
      <div id="player"></div>
    </div>
    <div id="loader" hidden>
      <div id="load-circle"></div>
    </div>
    <div id="video-info" class="d-flex flex-column align-items-start py-3 px-2 w-100"></div>
  </div>

  <div class="border rounded-4 d-flex flex-column align-items-start w-100" style="height: 600px; ">
    <h3 id="playlistTitle" class="px-4 py-3 w-100 border-bottom fs-1 d-flex align-items-start"></h3>
    <ul id="videoList" class="list-unstyled d-flex flex-column gap-1 ps-1 py-1" style="overflow-y: auto; flex-grow: 1;">
</ul>
  </div>

  <script type="module" src="func.js"></script>
```

```
  <script src="https://www.youtube.com/iframe_api"></script>
  <script type="module" src="load.js"></script>
  <script type="module" src="sort.js"></script>
  <script type="module" src="event.js"></script>
</body>
</html>
```

style.js

```
#videoList li{
    display: flex;
    border-radius: 0.75rem;
}

#videoList li>span{
    width: 2.3rem;
    text-align: center;
    align-self: center;
    color: #666;
}

#videoList li div{
    display: flex;
    flex-direction: column;
    align-items: start;
    padding: 2px 0 0 0.625rem;
    width: 67%;
    font-size: 1rem;
    font-weight: bold;
}

#videoList li div span{
    line-height: 1.2rem;
    max-height: 2.4rem;
    overflow: hidden;
}

#videoList .channel-title-and-view-count{
    font-size: 0.75rem;
}

.channel-title-and-view-count{
    color: #666;
    margin-top: 0.375rem;
}

#videoList li.current{
    background-color: #e0e8f5;
}

#video-info span:first-child{
    font-weight: bold;
    font-size: 1.25rem;
}

#videoList img{
    object-fit: cover;
    object-position: center;
    width: 23%;
    border-radius: 0.375rem;
}

body{
    max-width:800px;
}
```

```css
button{
    border: solid 1px black;
}

.form-control {
    border-color: #999;
}

#player{
    display: flex;
    justify-content: center;
    align-items: center;
    width: 100%;
    height: 100%;
    border-radius: 0.375rem;
}

#loader {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    z-index: 10;
    background-color: white;
    width: 105%;
    height: 105%;
    display: flex;
    align-items: center;
    justify-content: center;
}

#load-circle {
    width: 6.25rem;
    height: 6.25rem;
    border-radius: 50%;
    border: 0.5rem solid rgba(0, 255, 255, 0.2);
    border-top-color: rgba(0, 255, 255, 1);
    animation: rotate 1s linear infinite;
}

.error{
    color: red;
}

@keyframes rotate {
    to {
        transform: rotate(360deg);
    }
}

@media (max-width: 600px) {
  html { font-size: 10px; }
  body {max-width: 100vw;}
  #videoList{
    padding-right: 2px;
  }
}

@media (hover: none) {
    .btn:hover{
        background-color: transparent;
    }
    .btn-outline-primary:hover{
        color: #0d6efd;
```

```
    }
    .btn-outline-dark:hover{
        color:black;
    }
}

@media (hover:hover){
    #videoList li:hover{
        background-color: #f0f0f0;
    }
}
```

load.js

```
import {format_view_count, parse_duration, sec_to_our_min } from "./func.js";
export let all_items=[];
export let sorted_items=[];
export let display_items=[];

//const CHANNEL_ID = "UCFz1nNoqzgfM-OhmhuI1fYg";
export const API_KEY = "AIzaSyC32k1f-L-HgWX7vZjHnPGTP-TnKa4eZtY";
const DEFAULT_PLAYLIST_ID = "PLg_Vllr2X7mKpLGkxsHPQe7NwIyLZ6fF5"
let player;
let idx=0;
let ever_played=false; //再生リストを一度でも再生したか(create_list()でリセット)
let params = new URLSearchParams(location.search);
let playlist_id = params.get("Id") || DEFAULT_PLAYLIST_ID;
const cur_url = new URL(window.location);

const video_list=document.getElementById("videoList");

export const create_list=()=>{
    video_list.innerHTML="";
    display_items.forEach((item,i)=>{
        const li=document.createElement("li");
        li.dataset.index=i;
        const div=document.createElement("div");
        const title=document.createElement("span");
        const channel_name_and_view_count=document.createElement("span");
        const img=document.createElement("img");
        const index=document.createElement("span");

        title.textContent=item.snippet.title;
        channel_name_and_view_count.textContent=`${item.snippet.videoOwnerChannelTitle}・
${format_view_count(item.viewCount)}・${sec_to_our_min(item.duration)}`;
        channel_name_and_view_count.classList.add("channel-title-and-view-count");
        div.appendChild(title)
        div.appendChild(channel_name_and_view_count)
        img.src=item.snippet?.thumbnails?.medium?.url ?? null;
        index.textContent=i+1;
        li.appendChild(index);
        li.appendChild(img);
        li.appendChild(div);
        video_list.appendChild(li);

        li.addEventListener("click", () => {
            idx = parseInt(li.dataset.index);
            loadVideo();
        });
    });
    idx = 0;
    if(player && display_items.length > 0){
        loadVideo(true);
    }
    ever_played=false;
```

```
    video_list.scrollTo({
        top: 0
    });
}

export const reset=()=>{
    document.getElementById("playlistTotalTime").innerHTML="";
    document.getElementById("createPlaylistTime").value="";
    document.getElementById("minViews").value="";
    document.getElementById("maxViews").value="";
    document.getElementById("sort").value="default";
}

const loader=document.getElementById("loader");
export const load=async(playlist_id)=> {
    //playlistの概要情報の取得
    const id_error=document.getElementById("idError");
    let playlist_data=null;
    try {
        const playlist_url =`https://www.googleapis.com/youtube/v3/playlists?
part=snippet&id=${playlist_id}&key=${API_KEY}`;
        const playlist_res = await fetch(playlist_url);
        playlist_data = await playlist_res.json();
        console.log(playlist_data)
    } catch (error) {
        id_error.hidden=false;
        return;
    }

    if(!playlist_data.items || playlist_data.items.length===0){
        id_error.hidden=false;
        return;
    }else{
        id_error.hidden=true;
    }
    loader.hidden=false;

    //URLとhistoryの処理
    await new Promise(requestAnimationFrame);
    document.getElementById("playlistId").value=playlist_id;
    if(playlist_id!==DEFAULT_PLAYLIST_ID && cur_url.searchParams.get("Id")!==playlist_id){
        cur_url.searchParams.set("Id",playlist_id);
        history.pushState(null,"",cur_url);
    }

    reset();

    document.getElementById("playlistTitle").textContent=playlist_data.items[0].snippet.title;
    //キャッシュ処理
    const cache = JSON.parse(localStorage.getItem(playlist_id));
    if(cache && cache.etag===playlist_data.etag){
        all_items = cache.all_items;
    }else{
        video_list.innerHTML="";
        //playlistの動画取得
        let next_page_token ="";
        const max_results=50;
        const max_loop=500/max_results;
        all_items=[];
        let loop_i=0;
        do{
            const url=`https://www.googleapis.com/youtube/v3/playlistItems?
part=snippet&playlistId=${playlist_id}&maxResults=${max_results}&key=${API_KEY}`+
                (next_page_token ? `&pageToken=${next_page_token}`:"");
            const res = await fetch(url);
```

```
            const data = await res.json();
            all_items.push( ... data.items);
            next_page_token = data.nextPageToken;
            loop_i++;
        }while(next_page_token && loop_i<max_loop);
        let all_video_ids=all_items.map(item=>item.snippet.resourceId.videoId)

        const chunk_size=50;
        let video_datas=[]
        for(let i=0;i<all_video_ids.length;i+=chunk_size){
            const chunk=all_video_ids.slice(i,i+chunk_size).join(",");
            const url=`https://www.googleapis.com/youtube/v3/videos?
part=contentDetails,statistics&id=${chunk}&key=${API_KEY}`
            const res=await fetch(url);
            const data =await res.json();
            video_datas.push( ... data.items);
        }

        let j=0;
        for(let i=0;i<all_items.length;i++){
            const item=all_items[i];
            const title = item.snippet.title;
            const thumbs = item.snippet.thumbnails;
            if(!thumbs || ["Deleted video", "Private video", "Unavailable video"].includes(title)){
                item.duration=0;
                j++;
            }else{
                item.viewCount=Number(video_datas[i-j].statistics.viewCount);
                item.duration=parse_duration(video_datas[i-j].contentDetails.duration);
            }
        }
    }

    sorted_items=[ ... all_items];
    display_items=[ ... all_items];
    localStorage.setItem(playlist_id, JSON.stringify({etag:playlist_data.etag,all_items:all_items}));
    create_list();
    console.log(display_items)
}

const highLightCurrentVideo = () => {
    const videoList = document.getElementById("videoList");
    const lis = videoList.querySelectorAll("li");
    lis.forEach((li, i) => {
        const videoIndex = li.querySelector("span");
        const isCurrent = i === idx;
        li.classList.toggle("current", isCurrent);
        videoIndex.textContent = isCurrent ? "▶" : (i + 1);
    });

    const currentLi = videoList.querySelector("li.current");
    const liRect = currentLi.getBoundingClientRect();
    const ulRect = videoList.getBoundingClientRect();
    if (currentLi) {
        videoList.scrollTo({
            top: videoList.scrollTop + (liRect.top - ulRect.top)-1,
            behavior: 'smooth'
        });
    }
};

const video_info=document.getElementById("video-info");
function updateVideoInfo() {
    video_info.innerHTML = "";
```

```
    const title = document.createElement("span");
    const channel_name_and_view_count = document.createElement("span");
    channel_name_and_view_count.classList.add("channel-title-and-view-count");

    title.textContent = display_items[idx].snippet.title;
    channel_name_and_view_count.textContent = `${display_items[idx].snippet.videoOwnerChannelTitle}・
${format_view_count(display_items[idx].viewCount)}`;

    video_info.appendChild(title);
    video_info.appendChild(channel_name_and_view_count);
}

function loadVideo(isCue=false) {
    if(isCue){
        const videoList = document.getElementById("videoList");
        const currentLi = videoList.querySelector("li.current");
        if(currentLi){
            currentLi.classList.toggle("current");
            currentLi.querySelector("span").textContent = idx;
        }
        player.cueVideoById(display_items[idx].snippet.resourceId.videoId);
    }else{
        player.loadVideoById(display_items[idx].snippet.resourceId.videoId);
        highLightCurrentVideo();
    }
    updateVideoInfo();
}

const initPlayer=()=>{
    if(!display_items || display_items.length===0){
        console.error("動画データが揃っていません");
        return;
    }

    function onPlayerError(event){
        console.warn("動画再生エラー:", event.data, "idx:", idx);
        // エラーが出たら次の動画へ
        idx++;
        if(idx < display_items.length){
            if(ever_played){
                loadVideo();
            }else{
                loadVideo(true);
            }
        }
    }

    function onPlayerStateChange(event){
        switch(event.data){
        case YT.PlayerState.PLAYING:
            highLightCurrentVideo();
            ever_played=true;
            break;
        case YT.PlayerState.ENDED:
            idx++;
            if(idx<display_items.length){
                loadVideo();
            }else{
                console.log("全部再生しました");
            }
            break;
        case YT.PlayerState.CUED:
            loader.hidden=true;
            break;
        }
```

```
    }

    player=new YT.Player('player',{
        width:640,
        height:360,
        videoId:display_items[idx].snippet.resourceId.videoId,
        playerVars:{
            autoplay:0,
            controls:1,
        },
        events:{
            'onReady':()=>loader.hidden=true,
            'onStateChange':onPlayerStateChange,
            'onError':onPlayerError
        }
    });
}

// IFrame API が ready になったら initPlayer を呼ぶ
const waitForYouTubeAPI = () => {
    return new Promise(resolve => {
        if (window.YT && YT.Player) resolve();
        else window.onYouTubeIframeAPIReady = () => resolve();
    });
}

const initApp = async () => {
    await load(playlist_id);
    await waitForYouTubeAPI();
    initPlayer();
    updateVideoInfo();
}

initApp();

window.addEventListener("popstate", () => {
    params = new URLSearchParams(location.search);
    playlist_id = params.get("Id") || DEFAULT_PLAYLIST_ID;
    load(playlist_id);
});
```

event.js

```
import { sec_to_our_min, shuffle_array } from "./func.js";
import { create_list, all_items,display_items, load, reset, sorted_items } from "./load.js";
import { compare_position_at,compare_view_count,sort,dp} from "./algorithm.js";

const handle_sort=(event)=>{
    const value=event.target.value;
    switch (value) {
    case "default":
        sorted_items.splice(0,display_items.length, ... all_items);
        display_items.splice(0,display_items.length, ... all_items);
        break;
    case "positionDesc":
        sort(sorted_items,compare_position_at);
        sort(display_items, compare_position_at);
        break;
    case "positionAsc":
        sort(sorted_items,compare_position_at,true);
        sort(display_items, compare_position_at, true);
        break;
    case "viewCountDesc":
        sort(sorted_items,compare_view_count);
        sort(display_items, compare_view_count);
```

```
        break;
    case "viewCountAsc":
        sort(sorted_items,compare_view_count,true);
        sort(display_items, compare_view_count, true);
        break;
    }
    create_list();
}

document.getElementById("sort").addEventListener("change",handle_sort);


const search_form=document.getElementById("searchForm");

const handle_search=(event)=>{
    event.preventDefault();
    const data = new FormData(search_form);

    const minViews=Number(data.get("minViews"))*10000;
    const maxViews=data.get("maxViews")=='' ? 100000000000:Number(data.get("maxViews"))*10000;

    const from_date = data.get("fromDate");
    const to_date   = data.get("toDate");
    const filterd_items=sorted_items.filter(item=>{
        const view_bool=minViews<=item.viewCount && item.viewCount<=maxViews;

        const from_bool = from_date ? new Date(item.snippet.publishedAt) >= new Date(from_date) : true;
        const to_bool = to_date ? new Date(item.snippet.publishedAt) <= new Date(to_date) : true;
        return view_bool && from_bool && to_bool;
    });

    display_items.splice(0,display_items.length, ... filterd_items);
    create_list();
    document.getElementById("playlistTotalTime").innerHTML="";
    document.getElementById("createPlaylistTime").value="";
}

search_form.addEventListener("submit",handle_search);


const handle_create_playlist=(event) => {
    event.preventDefault();
    const target_minutes = Number(document.getElementById("createPlaylistTime").value);
    const times_arr=sorted_items.map(item=>{
        return item.duration;
    })
    const dp_res=dp(times_arr,target_minutes);
    console.log(dp_res);
    const new_play_list=dp_res.indexes.map(i=>sorted_items[i]);
    display_items.splice(0,display_items.length, ... new_play_list);
    create_list();
    document.getElementById("playlistTotalTime").innerHTML=`${sec_to_our_min(dp_res.best_sum)} のプレイリストを作成しました!
`;
    document.getElementById("minViews").value="";
    document.getElementById("maxViews").value="";
}

document.getElementById("createPlaylistForm").addEventListener("submit",handle_create_playlist );


document.getElementById("reset").addEventListener("click",()=>{
    sorted_items.splice(0,sorted_items.length, ... all_items);
    display_items.splice(0,display_items.length, ... all_items);
    reset();
    create_list();
```

```
});

document.getElementById("shuffle").addEventListener("click",()=>{
    const shuffle_arr=shuffle_array(all_items);
    sorted_items.splice(0,sorted_items.length, ... shuffle_arr);
    display_items.splice(0,display_items.length, ... shuffle_arr);
    create_list();
});

document.getElementById("playlistIdForm").addEventListener("submit",async(event)=>{
    event.preventDefault();
    const playlist_id=document.getElementById("playlistId").value;
    load(playlist_id);
});
```

algorithm.js

```
export const compare_publishd_at=(playlist_item_a,play_list_item_b)=>{
    const a=playlist_item_a.snippet.publishedAt;
    const b=play_list_item_b.snippet.publishedAt;
    return new Date(a)>new Date(b);
};

export const compare_position_at=(playlist_item_a,play_list_item_b)=>{
    const a=playlist_item_a.snippet.position;
    const b=play_list_item_b.snippet.position;
    return a>b;
};

export const compare_view_count=(play_list_item_a,play_list_item_b)=>{
    const a=play_list_item_a.viewCount;
    const b=play_list_item_b.viewCount;
    return a>b;
}

export const sort=(array,compare_func,reverse=false)=>{
    for (let i=0;i<array.length-1;i++){
        for(let j=0;j<array.length-1-i;j++){
            if(
                reverse ?
                compare_func(array[j],array[j+1])
                :
                !compare_func(array[j],array[j+1])
            ){
                let tmp=array[j+1];
                array[j+1]=array[j];
                array[j]=tmp;
            }
        }
    }
    return array;
};

export const dp = (arr, target) => {
    if (arr.length === 0) {
        return {
            best_sum: 0,
            indexes: []
        };
    }

    target *= 60;
    const t_max = target + Math.max( ... arr);

    let can = Array(t_max + 1).fill(false);
```

```
    let prev = Array.from({ length: t_max + 1 }, () => []); // 複数候補を持つ
    can[0] = true;

    // DPで作れる値をマッピング
    for (let i = 0; i < arr.length; i++) {
        const time = arr[i];
        if(time==0) continue; //動画時間がゼロ(消されたビデオなど)はいれない
        //target以下の値全てで試す
        //time以外の値でj-timeが作れるなら,jも作れる
        for (let j = t_max; j >= time; j--) {
            if (can[j - time]) {
                can[j] = true;
                prev[j].push(i); // 複数候補を追加
            }
        }
    }

    // bestの探索（targetに最も近い値）
    let best = 0;
    for (let i = t_max; i >= 0; i--) {
        if (can[i] && Math.abs(target - i) < Math.abs(target - best)) {
            best = i;
        }
    }

    // bestな組み合わせの中から一つをランダムに復元
    let res = [];
    let now = best;
    while (now > 0) {
        const prev_now_arr = prev[now];
        const prev_now = prev_now_arr[Math.floor(Math.random()*prev_now_arr.length)];
        res.push(prev_now);
        now -= arr[prev_now];
    }

    return {
        best_sum: best,
        indexes: res
    };
};
```

func.js

```
export const format_view_count=(num)=> {
  if(!num) return '';
  if (num >= 100000000) {
    const value = (num / 100000000).toFixed(1);
    return value.replace(/\.0$/, '') + '億 回視聴';
  } else if (num >= 10000) {
    return Math.floor(num / 10000) + '万 回視聴';
  } else {
    return num + ' 回視聴';
  }
}

export const parse_duration=(iso)=> {
  const match = iso.match(/PT(?:(\d+)H)?(?:(\d+)M)?(?:(\d+)S)?/);
  if (!match) return 0;
  const hours = parseInt(match[1] || 0);
  const minutes = parseInt(match[2] || 0);
  const seconds = parseInt(match[3] || 0);
  return hours * 3600 + minutes * 60 + seconds;
}
```

```
export const sec_to_our_min = (sec) => {
  const hours = Math.floor(sec / 3600);
  const mins = Math.floor((sec % 3600) / 60);
  const secs = sec % 60;

  let text = "";

  if (hours > 0) text += `${hours}時間`;
  if (mins > 0) text += `${mins}分`;
  if (secs > 0) text += `${secs}秒`;

  if (text === "") text = "0秒";

  return text;
};

export const shuffle_array=(array)=> {
  const copy = array.slice(); // 元の配列はコピーして保持
  for (let i = copy.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [copy[i], copy[j]] = [copy[j], copy[i]]; // 要素を入れ替え
  }
  return copy;
}
```